

LNAI 13385

Jasmin Blanchette  
Laura Kovács  
Dirk Pattinson (Eds.)

# Automated Reasoning

11th International Joint Conference, IJCAR 2022  
Haifa, Israel, August 8–10, 2022  
Proceedings



# **Lecture Notes in Artificial Intelligence**

**13385**

Subseries of Lecture Notes in Computer Science

## Series Editors

Randy Goebel

*University of Alberta, Edmonton, Canada*

Wolfgang Wahlster

*DFKI, Berlin, Germany*

Zhi-Hua Zhou

*Nanjing University, Nanjing, China*

## Founding Editor

Jörg Siekmann

*DFKI and Saarland University, Saarbrücken, Germany*

More information about this subseries at <https://link.springer.com/bookseries/1244>


Jasmin Blanchette · Laura Kovács ·  
Dirk Pattinson (Eds.)

# Automated Reasoning

11th International Joint Conference, IJCAR 2022  
Haifa, Israel, August 8–10, 2022  
Proceedings



### Editors

Jasmin Blanchette   
Vrije Universiteit Amsterdam  
Amsterdam, The Netherlands

Laura Kovács   
Vienna University of Technology  
Wien, Austria

Dirk Pattinson   
Australian National University  
Canberra, ACT, Australia



ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Artificial Intelligence

ISBN 978-3-031-10768-9

ISBN 978-3-031-10769-6 (eBook)

<https://doi.org/10.1007/978-3-031-10769-6>

LNCS Sublibrary: SL7 – Artificial Intelligence

© The Editor(s) (if applicable) and The Author(s) 2022. This book is an open access publication.

**Open Access** This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

This volume contains the papers presented at the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) held during August 8–10, 2022, in Haifa, Israel. IJCAR was part of the Federated Logic Conference (FLoC 2022), which took place from July 31 to August 12, 2022, in Haifa.

IJCAR is the premier international joint conference on all aspects of automated reasoning, including foundations, implementations, and applications, comprising several leading conferences and workshops. IJCAR 2022 united the Conference on Automated Deduction (CADE), the International Symposium on Frontiers of Combining Systems (FroCoS), and the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX). Previous IJCAR conferences were held in Siena, Italy, in 2001, Cork, Ireland, in 2004, Seattle, USA, in 2006, Sydney, Australia, in 2008, Edinburgh, UK, in 2010, Manchester, UK, in 2012, Vienna, Austria, in 2014, Coimbra, Portugal, in 2016, Oxford, UK, in 2018, and Paris, France, in 2020 (virtual).

There were 85 submissions. Each submission was assigned to at least three Program Committee members and was reviewed in single-blind mode. The committee decided to accept 41 papers: 32 regular papers and nine system descriptions.

The program also included two invited talks, by Elvira Albert and Gilles Dowek, as well as a plenary FLoC talk by Aarti Gupta.

We acknowledge the FLoC sponsors:

- Diamond sponsors: Amazon Web Services, Meta, Intel
- Gold sponsors: Google, Nvidia, Synopsys
- Silver sponsor: Cadence
- Bronze sponsors: DLVSystem, Veridise
- Other sponsors: Technion, The Henry and Marilyn Taub Faculty of Computer Science

We also acknowledge the generous sponsorship of Springer and the Trakhtenbrot family, as well as the invaluable support provided by the EasyChair developers. We finally thank the FLoC 2022 organization team for assisting us with local organization and general conference management.

May 2022

Jasmin Blanchette  
Laura Kovács  
Dirk Pattinson

# Organization

## Program Committee

Erika Abraham	RWTH Aachen University, Germany
Carlos Areces	Universidad Nacional de Córdoba, Spain
Bernhard Beckert	Karlsruhe Institute of Technology, Germany
Alexander Bentkamp	Chinese Academy of Sciences, China
Armin Biere	University of Freiburg, Germany
Nikolaj Bjørner	Microsoft, USA
Jasmin Blanchette (Co-chair)	Vrije Universiteit Amsterdam, The Netherlands
Frédéric Blanqui	Inria, France
Maria Paola Bonacina	Università degli Studi di Verona, Italy
Kaustuv Chaudhuri	Inria, France
Agata Ciabattoni	Vienna University of Technology, Austria
Stéphane Demri	CNRS, LMF, ENS Paris-Saclay, France
Clare Dixon	University of Manchester, UK
Huimin Dong	Sun Yat-sen University, China
Katalin Fazekas	Vienna University of Technology, Austria
Mathias Fleury	University of Freiburg, Austria
Pascal Fontaine	Université de Liège, Belgium
Nathan Fulton	IBM, USA
Silvio Ghilardi	Università degli Studi di Milano, Italy
Jürgen Giesl	RWTH Aachen University, Germany
Rajeev Gore	Australian National University, Australia
Marijn Heule	Carnegie Mellon University, USA
Radu Iosif	Verimag, CNRS, Université Grenoble Alpes, France
Mikolas Janota	Czech Technical University in Prague, Czech Republic
Moa Johansson	Chalmers University of Technology, Sweden
Cezary Kaliszyk	University of Innsbruck, Austria
Laura Kovacs (Co-chair)	Vienna University of Technology, Austria
Orna Kupferman	Hebrew University, Israel
Cláudia Nalon	University of Brasília, Brazil
Vivek Nigam	Huawei ERC, Germany
Tobias Nipkow	Technical University of Munich, Germany
Jens Otten	University of Oslo, Norway
Dirk Pattinson (Co-chair)	Australian National University, Australia
Nicolas Peltier	CNRS, LIG, France

Brigitte Pientka	McGill University, Canada
Elaine Pimentel	University College London, UK
André Platzner	Carnegie Mellon University, USA
Giles Reger	Amazon Web Services, USA, and University of Manchester, UK
Andrew Reynolds	University of Iowa, USA
Simon Robillard	Université de Montpellier, France
Albert Rubio	Universidad Complutense de Madrid, Spain
Philipp Ruegger	Uppsala University, Sweden
Renate A. Schmidt	University of Manchester, UK
Stephan Schulz	DHBW Stuttgart, Germany
Roberto Sebastiani	University of Trento, Italy
Martina Seidl	Johannes Kepler University Linz, Austria
Viorica Sofronie-Stokkermans	University of Koblenz-Landau, Germany
Lutz Straßburger	Inria, France
Martin Suda	Czech Technical University in Prague, Czech Republic
Tanel Tammet	Tallinn University of Technology, Estonia
Sophie Tourret	Inria, France, and Max Planck Institute for Informatics, Germany
Uwe Waldmann	Max Planck Institute for Informatics, Germany
Christoph Weidenbach	Max Planck Institute for Informatics, Germany
Sarah Winkler	Free University of Bozen-Bolzano, Italy
Yoni Zohar	Bar-Ilan University, Israel

## Additional Reviewers

László Antal	Samir Genaim
Paolo Baldi	Alessandro Gianola
Lionel Blatter	Raúl Gutiérrez
Brandon Bohrer	Fajar Haifani
Marius Bozga	Alejandro Hernández-Cerezo
Chad Brown	Ullrich Hustadt
Lucas Bueri	Jan Jakubuv
Guillaume Burel	Martin Jonas
Marcelo Coniglio	Michael Kirsten
Riccardo De Masellis	Gereon Kremer
Warren Del-Pinto	Roman Kuznets
Zafer Esen	Jonathan Laurent
Michael Färber	Chencheng Liang
Sicun Gao	Enrico Lipparini
Jacques Garrigue	Florin Manea
Thibault Gauthier	Marco Maratea

Sonia Marin  
Enrique Martin-Martin  
Andrea Mazzullo  
Stephan Merz  
Antoine Miné  
Sibylle Möhle  
Cristian Molinaro  
Markus Müller-Olm  
Jasper Nalbach  
Joel Ouaknine  
Tobias Paxian  
Wolfram Pfeifer  
Andrew Pitts

Amaury Pouly  
Stanisław Purgał  
Michael Rawson  
Giselle Reis  
Clara Rodríguez-Núñez  
Daniel Skurt  
Giuseppe Spallitta  
Sorin Stratulat  
Petar Vukmirović  
Alexander Weigl  
Richard Zach  
Anna Zamansky  
Michał Zawidzki

# Contents

## Invited Talks

Using Automated Reasoning Techniques for Enhancing the Efficiency and Security of (Ethereum) Smart Contracts .....	3
<i>Elvira Albert, Pablo Gordillo, Alejandro Hernández-Cerezo, Clara Rodríguez-Núñez, and Albert Rubio</i>	

From the Universality of Mathematical Truth to the Interoperability of Proof Systems .....	8
<i>Gilles Dowek</i>	

## Satisfiability, SMT Solving, and Arithmetic

Flexible Proof Production in an Industrial-Strength SMT Solver .....	15
<i>Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark Barrett</i>	

CTL* Model Checking for Data-Aware Dynamic Systems with Arithmetic ....	36
<i>Paolo Felli, Marco Montali, and Sarah Winkler</i>	

SAT-Based Proof Search in Intermediate Propositional Logics .....	57
<i>Camillo Fiorentini and Mauro Ferrari</i>	

Clause Redundancy and Preprocessing in Maximum Satisfiability .....	75
<i>Hannes Ihalaainen, Jeremias Berg, and Matti Järvisalo</i>	

Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description) .....	95
<i>Gereon Kremer, Andrew Reynolds, Clark Barrett, and Cesare Tinelli</i>	

Preprocessing of Propagation Redundant Clauses .....	106
<i>Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant</i>	

Reasoning About Vectors Using an SMT Theory of Sequences .....	125
<i>Ying Sheng, Andres Nötzli, Andrew Reynolds, Yoni Zohar, David Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Clark Barrett, and Cesare Tinelli</i>	

## Calculi and Orderings

An Efficient Subsumption Test Pipeline for BS(LRA) Clauses .....	147
<i>Martin Bromberger, Lorenz Leutgeb, and Christoph Weidenbach</i>	
Ground Joinability and Connectedness in the Superposition Calculus .....	169
<i>André Duarte and Konstantin Korovin</i>	
Connection-Minimal Abduction in $\mathcal{EL}$ via Translation to FOL .....	188
<i>Fajar Haifani, Patrick Koopmann, Sophie Tourret, and Christoph Weidenbach</i>	
Semantic Relevance .....	208
<i>Fajar Haifani and Christoph Weidenbach</i>	
SCL(EQ): SCL for First-Order Logic with Equality .....	228
<i>Hendrik Leidingen and Christoph Weidenbach</i>	
Term Orderings for Non-reachability of (Conditional) Rewriting .....	248
<i>Akihisa Yamada</i>	

## Knowledge Representation and Justification

EVONNE: Interactive Proof Visualization for Description Logics (System Description) .....	271
<i>Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Raimund Dachzelt, Patrick Koopmann, and Julián Méndez</i>	
Actions over Core-Closed Knowledge Bases .....	281
<i>Claudia Cauli, Magdalena Ortiz, and Nir Piterman</i>	
GK: Implementing Full First Order Default Logic for Commonsense Reasoning (System Description) .....	300
<i>Tanel Tammet, Dirk Draheim, and Priit Järv</i>	
Hypergraph-Based Inference Rules for Computing $\mathcal{EL}^+$ -Ontology Justifications .....	310
<i>Hui Yang, Yue Ma, and Nicole Bidoit</i>	

## Choices, Invariance, Substitutions, and Formalizations

Sequent Calculi for Choice Logics .....	331
<i>Michael Bernreiter, Anela Lolic, Jan Maly, and Stefan Woltran</i>	

<b>Lash 1.0 (System Description)</b> .....	350
<i>Chad E. Brown and Cezary Kaliszyk</i>	
<b>Goéland: A Concurrent Tableau-Based Theorem Prover (System Description)</b> .....	359
<i>Julie Cailler, Johann Rosain, David Delahaye, Simon Robillard, and Hinde Lilia Bouziane</i>	
<b>Binary Codes that Do Not Preserve Primitivity</b> .....	369
<i>Štěpán Holub, Martin Raška, and Štěpán Starosta</i>	
<b>Formula Simplification via Invariance Detection by Algebraically Indexed Types</b> .....	388
<i>Takuya Matsuzaki and Tomohiro Fujita</i>	
<b>Synthetic Tableaux: Minimal Tableau Search Heuristics</b> .....	407
<i>Michał Sochański, Dorota Leszczyńska-Jasion, Szymon Chlebowski, Agata Tomczyk, and Marcin Jukiewicz</i>	
<b>Modal Logics</b>	
<b>Paraconsistent Gödel Modal Logic</b> .....	429
<i>Marta Bílková, Sabine Frittella, and Daniil Kozhemiachenko</i>	
<b>Non-associative, Non-commutative Multi-modal Linear Logic</b> .....	449
<i>Eben Blaisdell, Max Kanovich, Stepan L. Kuznetsov, Elaine Pimentel, and Andre Scedrov</i>	
<b>Effective Semantics for the Modal Logics K and KT via Non-deterministic Matrices</b> .....	468
<i>Ori Lahav and Yoni Zohar</i>	
<b>Local Reductions for the Modal Cube</b> .....	486
<i>Cláudia Nalon, Ullrich Hustadt, Fabio Papacchini, and Clare Dixon</i>	
<b>Proof Systems and Proof Search</b>	
<b>Cyclic Proofs, Hypersequents, and Transitive Closure Logic</b> .....	509
<i>Anupam Das and Marianna Girlando</i>	
<b>Equational Unification and Matching, and Symbolic Reachability Analysis in Maude 3.2 (System Description)</b> .....	529
<i>Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott</i>	



Leśniewski’s Ontology – Proof-Theoretic Characterization .....	541
<i>Andrzej Indrzejczak</i>	
Bayesian Ranking for Strategy Scheduling in Automated Theorem Provers ....	559
<i>Chaitanya Mangla, Sean B. Holden, and Lawrence C. Paulson</i>	
A Framework for Approximate Generalization in Quantitative Theories .....	578
<i>Temur Kutsia and Cleo Pau</i>	
Guiding an Automated Theorem Prover with Neural Rewriting .....	597
<i>Jelle Piepenbrock, Tom Heskes, Mikoláš Janota, and Josef Urban</i>	
Rensets and Renaming-Based Recursion for Syntax with Bindings .....	618
<i>Andrei Popescu</i>	
Finite Two-Dimensional Proof Systems for Non-finitely Axiomatizable Logics .....	640
<i>Vitor Greati and João Marcos</i>	
Vampire Getting Noisy: Will Random Bits Help Conquer Chaos? (System Description) .....	659
<i>Martin Suda</i>	
<b>Evolution, Termination, and Decision Problems</b>	
On Eventual Non-negativity and Positivity for the Weighted Sum of Powers of Matrices .....	671
<i>S. Akshay, Supratik Chakraborty, and Debtanu Pal</i>	
Decision Problems in a Logic for Reasoning About Reconfigurable Distributed Systems .....	691
<i>Marius Bozga, Lucas Bueri, and Radu Iosif</i>	
Proving Non-Termination and Lower Runtime Bounds with LoAT (System Description) .....	712
<i>Florian Frohn and Jürgen Giesl</i>	
Implicit Definitions with Differential Equations for KeYmaera X: (System Description) .....	723
<i>James Gallicchio, Yong Kiam Tan, Stefan Mitsch, and André Platzer</i>	

Automatic Complexity Analysis of Integer Programs via Triangular Weakly Non-Linear Loops .....	734
<i>Nils Lommen, Fabian Meyer, and Jürgen Giesl</i>	
<b>Author Index</b> .....	755

## **Invited Talks**



# Using Automated Reasoning Techniques for Enhancing the Efficiency and Security of (Ethereum) Smart Contracts

Elvira Albert<sup>1,2</sup>(✉) , Pablo Gordillo<sup>1</sup> , Alejandro Hernández-Cerezo<sup>1</sup> ,  
Clara Rodríguez-Núñez<sup>1</sup> , and Albert Rubio<sup>1,2</sup>

<sup>1</sup> Complutense University of Madrid, Madrid, Spain

<sup>2</sup> Instituto de Tecnología del Conocimiento, Madrid, Spain

elvira@fdi.ucm.es

The use of the Ethereum blockchain platform [17] has experienced an enormous growth since its very first transaction back in 2015 and, along with it, the verification and optimization of the programs executed in the blockchain (known as Ethereum *smart contracts*) have raised considerable interest within the research community. As for any other kind of programs, the main properties of smart contracts are their *efficiency* and *security*. However, in the context of the blockchain, these properties acquire even more relevance. As regards efficiency, due to the huge volume of transactions, the cost and response time of the Ethereum blockchain platform have increased notably: the processing capacity of the transactions is limited and it is providing low transaction ratios per minute together with increased costs per transaction. Ethereum is aware of such limitations and it is currently working on solutions to improve scalability with the goal of increasing its capacity. As regards security, due to the public nature and immutability of smart contracts and the fact that their public functions can be executed by any user at any time, programming errors can be exploited by attackers and have a high economic impact [7, 13]. Verification is key to ensure the security of smart contract's execution and provide safety guarantees. This talk will present our work on the use of automated reasoning techniques and tools to enhance the security and efficiency [2–4, 6] of Ethereum smart contracts along the two directions described below.

*Security.* Our main focus on security will be to detect and avoid potential *reentrancy* attacks, one of the best known and exploited vulnerabilities that have caused infamous attacks in the Ethereum ecosystem due to their economic impact [9, 11, 15]. Reentrancy attacks might occur on programs with callbacks, a mechanism that allows making calls among contracts. Callbacks occur when a method of a contract invokes a method of another contract and the latter, either directly or indirectly, invokes one or more methods of the former before the original method invocation returns. While this mechanism is useful and powerful

---

This work was funded partially by the Ethereum Foundation (Grant FY21-0372), the Spanish MCIU, AEI and FEDER (EU) project RTI2018-094403-B-C31 and by the CM project S2018/TCS-4314 co-funded by EIE Funds of the European Union.

© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 3–7, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_1](https://doi.org/10.1007/978-3-031-10769-6_1)

in event-driven programming, it has been used to exploit vulnerabilities. Our approach to detect potential reentrancy problems is to ensure that the program meets the Effectively Callback Freeness (ECF) property [10]. ECF guarantees the modularity of a contract in the sense that executions with callbacks cannot result in new states that are not reachable by callback free executions. This implies that the use of callbacks will not lead to unpredicted, potentially dangerous, states. In order to ensure the ECF property, we use commutation and projection of fragments of code [6]. Intuitively, given a function fragment  $A$  followed by  $B$  (denoted  $A.B$ ), in case we can receive a callback to some function  $f$  between these fragments (that is,  $A.f.B$ ), we ensure safety by proving that this execution that contains callbacks is equivalent to a callback free execution: either to  $A.B$  (projection),  $f.A.B$  (left-commutation) or  $A.B.f$  (right-commutation). The use of automated reasoning techniques enables proving this kind of properties. Inspired by the use of SMT solvers to prove redundancy of concurrent executions [1, 8, 16], we have implemented such checks using state-of-the-art SMT solvers.

The ECF property can be generalized to allow callbacks to introduce new behaviors as long as they are benign, as [5] does by defining the notion of R-ECF. The main difference between ECF and R-ECF is that while ECF checks that the states reached by executions with callbacks are exactly the same as the ones reached by executions that do not contain callbacks, R-ECF checks that they satisfy a relation with respect to the states reached without callbacks. This way, R-ECF is able to recognize and distinguish the benign behaviors introduced by callbacks from the ones that are potentially dangerous, while ECF cannot. The main application of R-ECF is that, from a particular invariant of the program, it allows reducing the problem of verifying the invariant in the presence of callbacks, to the callback-free setting. For example, if we consider the invariant  $\text{balance} \geq 0$  and prove that the contract is R-ECF with respect to the relation  $\text{balance}_{cb} \geq \text{balance}_{cbfree}$  (i.e., the balance reached by executions with callbacks is greater than the one reached without callbacks), then we only need to consider callback free executions in order to prove the preservation of the invariant.

We considered as benchmarks the top-150 contracts based on volume of usage, and studied the modularity of their functions in terms of ECF and R-ECF. A total of 386 of their functions were susceptible to have callbacks, from which 62.7% were verified to be ECF. The R-ECF approach was able to increase the accuracy of the analysis, being able to prove the correctness of an extra 2% of functions [5, 6].

*Efficiency.* The main focus on efficiency will be on optimizing the resource consumption of smart contract executions. On the Ethereum blockchain, the resource consumption is measured in terms of *gas*, a unit introduced in the system to quantify the computational effort and charge a fee accordingly in order to have a transaction executed. To understand how we can optimize gas, we need to discuss it (and do it) at the level of the Ethereum bytecode. Smart contracts in Ethereum are executed using the Ethereum Virtual Machine (EVM). The EVM is a simple stack-based architecture which uses 256-bit words and has its own repertory of instructions (EVM opcodes). In the EVM, the mem-

ory model is split into two different structures: the *storage*, which is persistent between transactions and expensive to use; and the *memory*, which does not persist between transactions and is cheaper. Each opcode has a gas cost associated to its execution. Besides, an additional fee must be paid for each byte when the smart contract is deployed. Thus, the resource to be optimized can be either the total amount of gas in a program or its size. Even though both criteria are usually related, there are some situations in which they do not correlate. For instance, pushing a big number in the stack consumes a small amount of gas and increases significantly the bytecode size, whereas obtaining the same value using arithmetic operations is more expensive but involves fewer bytes.

Among all possible techniques to optimize code, we have used the technique known as superoptimization [12]. The main idea of superoptimization is automatically finding an equivalent optimal sequence of instructions to another given loop-free sequence. In order to achieve this goal, we enumerate all possible candidates and determine the best option among them *wrt.* the optimization criteria. In the context of EVM, there exists several superoptimizers: EBSO [14], SYRUP [3, 4] and GASOL [2]. The techniques presented in this work correspond to the ones implemented in GASOL, which are an improvement and extension of the ones in SYRUP. We apply two kinds of automated reasoning techniques to superoptimize Ethereum smart contracts, symbolic execution and Max-SMT as described next.

- Symbolic execution is used to obtain a representation on how the stack and memory evolves *wrt.* to an initial stack. We determine the lowest size of the stack needed to perform all the operations in a block and apply symbolic execution to an initial stack containing that number of unknown stack variables. Opcodes representing operations that don't manage the stack are left as uninterpreted functions. Then, we apply as many simplification rules as possible from a fixed set of rules. Depending on the chosen criteria, some rules are disabled if they lead to worse candidates. Moreover, we apply static analysis regarding memory opcodes to determine whether there are some redundant store or load operations inside a block that can be safely removed or replaced. This leads to a simplified specification of the optimal block.
- The second technique involves synthesizing the optimal block from a given symbolic representation using a Max-SMT solver. The synthesis problem is expressed as a first-order formula in which every model corresponds to a valid equivalent block. Our encoding is expressed in the simple logic *QF\_IDL*, so that the Max-SMT solver can reason effectively on EVM blocks. In this encoding, the length of the sequence of instructions is fixed by an upper bound so that quantifiers are avoided. NOP operations are considered in the encoding to allow shorter sequences. The state of the stack is represented explicitly for each position in the sequence. Every instruction in the block and every basic stack operation have a constraint that reflects the impact they have on the stack for each possible position. Memory accesses are encoded as a partial order relation that synthesizes the dependencies among them. Regarding the optimization process, we express the cost (gas or bytes-size) of

each instruction using soft constraints. For both criteria, the corresponding set of soft constraints satisfies that an optimal model returned by the solver corresponds to an optimal block for that criteria.

Combining both approaches, we obtain significant savings for both criteria. For a subset of 30 smart contracts, selected among the latest published in Etherscan as of June 21, 2021 and optimized using the compiler solc v0.8.9, GASOL still manages to reduce 0.72% the amount of gas with the gas criteria enabled, and decreases the overall size by 3.28% with the size criteria enabled.

*Future work.* The current directions for future work include enhancing the performance of the smart contract optimizer in both accuracy and scalability of the process while keeping the efficiency. For the accuracy we are currently working on adding further reasoning on non-stack operations while staying in a quite simple logic. This will allow us to consider a wider set of equivalent blocks and hence increase the savings. Scalability can be threatened when we consider blocks of code of large size. We are investigating different approaches to scale better, including heuristics to partition the blocks in smaller sub-blocks, more efficient SMT encodings, among others. Finally, another direction for future work is to formally prove the correctness of the optimizer, *i.e.* developing a checker that can formally prove the equivalence of the optimized and the original (Ethereum) bytecode. For this, we are planning to use the Coq proof assistant in which we will develop a checker that, given an original bytecode –that corresponds a block of the control flow graph– and its optimization, it can formally prove their equivalence for any possible execution, and optionally it can generate a soundness proof that can be used as certificate.

## References

1. Albert, E., Gómez-Zamalloa, M., Isabel, M., Rubio, A.: Constrained dynamic partial order reduction. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10982, pp. 392–410. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96142-2\\_24](https://doi.org/10.1007/978-3-319-96142-2_24)
2. Albert, E., Gordillo, P., Hernández-Cerezo, A., Rubio, A.: A Max-SMT superoptimizer for EVM handling memory and storage. In: Fisman, D., Rosu, G. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2022. LNCS, vol. 13243. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-99524-9\\_11](https://doi.org/10.1007/978-3-030-99524-9_11)
3. Albert, E., Gordillo, P., Hernández-Cerezo, A., Rubio, A., Schett, M.A.: Super-optimization of smart contracts. ACM Trans. Softw. Eng. Methodol. (2022)
4. Albert, E., Gordillo, P., Rubio, A., Schett, M.A.: Synthesis of super-optimized smart contracts using Max-SMT. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 177–200. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-53288-8\\_10](https://doi.org/10.1007/978-3-030-53288-8_10)
5. Albert, E., Grossman, S., Rinetzký, N., Nunez, C.R., Rubio, A., Sagiv, M.: Relaxed effective callback freedom: a parametric correctness condition for sequential modules with callbacks. IEEE Trans. Dependable Secure Comput. (2022)

6. Albert, E., Grossman, S., Rinetzky, N., Rodríguez-Núñez, C., Rubio, A., Sagiv, M.: Taming callbacks for smart contract modularity. In: Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2020, vol. 4, pp. 209:1–209:30 (2020)
7. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54455-6\\_8](https://doi.org/10.1007/978-3-662-54455-6_8)
8. Bansal, K., Koskinen, E., Tripp, O.: Automatic generation of precise and useful commutativity conditions. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 115–132. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-89960-2\\_7](https://doi.org/10.1007/978-3-319-89960-2_7)
9. Daian, P.: Analysis of the DAO exploit (2016). <http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/>
10. Grossman, S., et al.: Online detection of effectively callback free objects with applications to smart contracts. PACMPL, 2(POPL) (2018)
11. Liu, M.: Urgent: OUSD was hacked and there has been a loss of funds (2020). <https://medium.com/originprotocol/urgent-ousd-has-hacked-and-there-has-been-a-loss-of-funds-7b8c4a7d534c>. Accessed 29 Jan 2021
12. Massalin, H.: Superoptimizer - a look at the smallest program. In: Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), pp. 122–126 (1987)
13. Mehar, M.I., et al.: Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. J. Cases Inf. Technol. **21**(1), 19–32 (2019)
14. Nagele, J., Schett, M.A.: Blockchain superoptimizer. In: Proceedings of 29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR) (2019). <https://arxiv.org/abs/2005.05912>
15. Tarasov, A.: Millions lost: the top 19 DeFi cryptocurrency hacks of 2020 (2020). <https://cryptobriefing.com/50-million-lost-the-top-19-defi-cryptocurrency-hacks-2020/2>. Accessed 29 Jan 2021
16. Wang, C., Yang, Z., Kahlon, V., Gupta, A.: Peephole partial order reduction. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 382–396. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_29](https://doi.org/10.1007/978-3-540-78800-3_29)
17. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger (2019)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.







# From the Universality of Mathematical Truth to the Interoperability of Proof Systems

Gilles Dowek<sup>(✉)</sup>

Inria and ENS Paris-Saclay, Paris, France  
gilles.dowek@ens-paris-saclay.fr

## 1 Yet Another Crisis of the Universality of Mathematical Truth

The development of computerized proof systems, such as COQ, MATITA, AGDA, LEAN, HOL 4, HOL LIGHT, ISABELLE/HOL, MIZAR, etc. is a major step forward in the never ending quest of mathematical rigor. But it jeopardizes the universality of mathematical truth [5]: we used to have proofs of Fermat's little theorem, we now have COQ proofs of Fermat's little theorem, ISABELLE/HOL proofs of Fermat's little theorem, PVS proofs of Fermat's little theorem, etc. Each proof system: COQ, ISABELLE/HOL, PVS, etc. defining its own language for mathematical statements and its own truth conditions for these statements.

This crisis can be compared to previous ones, when mathematicians have disagreed on the truth of some mathematical statements: the discovery of the incommensurability of the diagonal and side of a square, the introduction of infinite series, the non-Euclidean geometries, the discovery of the independence of the axiom of choice, and the emergence of constructivity. All these past crises have been resolved.

## 2 Predicate Logic and Other Logical Frameworks

One way to resolve a crisis, such as that of non-Euclidean geometries, or that of the axiom of choice, is to view geometry, or set theory, as an axiomatic theory. The judgement that the statement *the sum of the angles in a triangle equals the straight angle* is true evolves to that that it is a consequence of the parallel axiom and of the other axioms of geometry. Thus, the truth conditions must be defined, not for the statements of geometry, but for arbitrary sequents: pairs  $\Gamma \vdash A$  formed with a theory, a set of axioms,  $\Gamma$  and a statement  $A$ .

This induces a separation between the definition of the truth conditions of a sequent: the logical framework and the definition of the various geometries as theories in this logical framework. This logical framework, Predicate logic, was made precise by Hilbert and Ackermann [13], in 1928, more than a century after the beginning of the crisis of non-Euclidean geometries. The invention of

Predicate Logic was a huge step forward. But Predicate Logic also has some limitations.

To overcome these limitation, it has been modernized in various ways in the last decades. First,  $\lambda$ -PROLOG [15] and ISABELLE [17] have extended Predicate logic with variable binding function symbols, such as the symbol  $\lambda$  in the term  $\lambda x. x$ . Then, the  $\lambda\Pi$ -calculus [12] has permitted to explicitly represent proof-trees, using the so-called Brouwer-Heyting-Kolmogorov algorithmic interpretation of proofs and Curry-de Bruijn-Howard correspondence. In a second stream of research, Deduction modulo theory [4, 6] has introduced a distinction between computation and deduction, in such a way that the statement  $27 \times 37 = 999$  computes to  $999 = 999$ , with the algorithm of multiplication, and then to  $\top$ , with the algorithm of natural number comparison. It thus has a trivial proof. A third stream of research has extended classical Predicate logic to an Ecumenical predicate logic [3, 9–11, 14, 18, 19] with both constructive and classical logical constants.

These streams of research have merged, to provide a logical framework, the  $\lambda\Pi$ -calculus modulo theory [2], also called Martin-Löf’s logical framework [16]. This framework permits function symbols to bind variables, it includes an explicit representation for proof-trees, it distinguishes computation from deduction, and it permits to define both constructive and classical logical constants. It is the basis of the language DEDUKTI, where Simple type theory, Martin-Löf’s type theory, the Calculus of constructions, etc. can easily be expressed.

### 3 The Theory $\mathcal{U}$

The expression in DEDUKTI of Simple type theory, Simple type theory with polymorphism, Simple type theory with predicate subtyping, the Calculus of constructions, etc. use symbol declarations and computation rules that play the *rôle* of axioms in Predicate logic. But, just like the various geometries or the various set theories share a lot of axioms and distinguish by a few, these theories share a lot of symbols and rules. This remark leads to defining a large theory, the theory  $\mathcal{U}$  [1], that contains Simple type theory, Simple type theory with polymorphism, Simple type theory with predicate subtyping, and the Calculus of constructions, etc. as sub-theories.

Many proofs developed in proof processing systems can be expressed in the theory  $\mathcal{U}$  and depending on the symbols and rules they use they can be translated to more common formulations of the theories implemented in these systems.

For instance, F. Thiré has expressed a large library of arithmetic, originally developed in MATITA, in an sub-theory of the theory  $\mathcal{U}$ , corresponding to Simple type theory with polymorphism and translated these proofs to the language of seven proof systems [20], Y. Gérard has expressed the first book of Euclid’s elements originally developed in COQ, in a sub-theory of the theory  $\mathcal{U}$ , corresponding to Predicate logic, and translated these proofs to the language of many proof systems, including predicate logic ones [8], and T. Felicissimo has shown that a large library of proofs originally developed in MATITA, including

a proof of Bertrand's postulate, could be expressed in predicative type theory and expressed in Agda [7].

## References

1. Blanqui, F., Dowek, G., Grienemberger, É., Hondet, G., Thiré, F.: Some axioms for mathematics. In: Kobayashi, N. (ed.) *Formal Structures for Computation and Deduction*, vol. 195, pp. 20:1–20:19. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
2. Cousineau, D., Dowek, G.: Embedding pure type systems in the lambda-pi-calculus modulo. In: Della Rocca, S.R. (ed.) *TLCA 2007*. LNCS, vol. 4583, pp. 102–117. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73228-0\\_9](https://doi.org/10.1007/978-3-540-73228-0_9)
3. Dowek, G.: On the definition of the classical connectives and quantifiers. In: Haeusler, E.H., de Campos Sanz, W., Lopes, B. (eds.) *Why is this a Proof?* Festschrift for Luiz Carlos Pereira. College Publications (2015)
4. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. *J. Autom. Reason.* **31**, 33–72 (2003). <https://doi.org/10.1023/A:1027357912519>
5. Dowek, G., Thiré, F.: The universality of mathematical truth jeopardized by the development of computerized proof systems. In: Arana, A., Pataut, F. (eds.) *Proofs*, To be published
6. Dowek, G., Werner, B.: Proof normalization modulo. *J. Symb. Log.* **68**(4), 1289–1316 (2003)
7. Felicissimo, T., Blanqui, F., Kumar Barnawal, A.: Predicativize: sharing proofs with predicative systems. Manuscript (2022)
8. Géran, Y.: *Mathématiques inversées de Coq. l'exemple de GeoCoq*. Master thesis (2021)
9. Gilbert, F.: *Extending higher-order logic with predicate subtyping: application to PVS. (Extension de la logique d'ordre supérieur avec le sous-typage par prédicats)*. PhD thesis, Sorbonne Paris Cité, France (2018)
10. Girard, J.-Y.: On the unity of logic. *Ann. Pure Appl. Logic* **59**(3), 201–217 (1993)
11. Grienemberger, É.: A logical system for an ecumenical formalization of mathematics. Manuscript (2020)
12. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. *J. ACM* **40**(1), 143–184 (1993)
13. Hilbert, D., Ackermann, W.: *Grundzüge der theoretischen Logik*. Springer-Verlag (1928)
14. Liang, C., Miller, D.: Unifying classical and intuitionistic logics for computational control. In: *28th Symposium on Logic in Computer Science*, pp. 283–292 (2013)
15. Miller, D., Nadathur, G.: *Programming with Higher-Order Logic*. Cambridge University Press (2012)
16. Nordström, B., Petersson, K., Smith, J.M.: *Programming in Martin-Löf's type theory*. Oxford University Press (1990)
17. Paulson, L.C.: Isabelle: the next 700 theorem provers. In: Odifreddi, P. (ed.) *Logic and Computer Science*, pp. 361–386. Academic Press (1990)
18. Pereira, L.C., Rodriguez, R.O.: Normalization, soundness and completeness for the propositional fragment of Prawitz'ecumenical system. *Rev. Port. Filos.* **73**(3–4), 1153–1168 (2017)
19. Prawitz, D.: Classical versus intuitionistic logic. In: Haeusler, E.H., de Campos Sanz, W., Lopes, B. (eds.) *Why is this a Proof?* Festschrift for Luiz Carlos Pereira. College Publications (2015)

20. Thiré, F.: Sharing a library between proof assistants: reaching out to the HOL family. In: Blanqui, F., Reis, G. (eds.) Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages, vol. 274, pp. 57–71. EPTCS (2018)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



# **Satisfiability, SMT Solving, and Arithmetic**



# Flexible Proof Production in an Industrial-Strength SMT Solver

Haniel Barbosa<sup>1</sup>, Andrew Reynolds<sup>2</sup>, Gereon Kremer<sup>3</sup>, Hanna Lachnitt<sup>3</sup>,  
Aina Niemetz<sup>3</sup>, Andres Nötzli<sup>3</sup>, Alex Ozdemir<sup>3</sup>, Mathias Preiner<sup>3</sup>,  
Arjun Viswanathan<sup>2</sup>, Scott Viteri<sup>3</sup>, Yoni Zohar<sup>4</sup>(✉), Cesare Tinelli<sup>2</sup>,  
and Clark Barrett<sup>3</sup>

<sup>1</sup> Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

<sup>2</sup> The University of Iowa, Iowa City, USA

<sup>3</sup> Stanford University, Stanford, USA

<sup>4</sup> Bar-Ilan University, Ramat Gan, Israel

yoniz206@gmail.com

**Abstract.** Proof production for SMT solvers is paramount to ensure their correctness independently from implementations, which are often prohibitively difficult to verify. Historically, however, SMT proof production has struggled with performance and coverage issues, resulting in the disabling of many crucial solving techniques and in coarse-grained (and thus hard to check) proofs. We present a flexible proof-production architecture designed to handle the complexity of versatile, industrial-strength SMT solvers and show how we leverage it to produce detailed proofs, including for components previously unsupported by any solver. The architecture allows proofs to be produced modularly, lazily, and with numerous safeguards for correctness. This architecture has been implemented in the state-of-the-art SMT solver *cvc5*. We evaluate its proofs for SMT-LIB benchmarks and show that the new architecture produces better coverage than previous approaches, has acceptable performance overhead, and supports detailed proofs for most solving components.

## 1 Introduction

SMT solvers [9] are widely used as backbones of formal methods tools in a variety of applications, often safety-critical ones. These tools rely on the solver’s correctness to guarantee the validity of their results such as, for instance, that an access policy does not inadvertently give access to sensitive data [4]. However, SMT solvers, particularly industrial-strength ones, are often extremely complex pieces of engineering. This makes it hard to ensure that implementation issues do not affect results. As the industrial use of SMT solvers increases, it is paramount to be able to convince non-experts of the trustworthiness of their results.

A solution is to decouple confidence from the implementation by coupling results with machine-checkable certificates of their correctness. For SMT solvers,

---

This work was partially supported by the Office of Naval Research (Contract No. 68335-17-C-0558), a gift from Amazon Web Services, and by NSF-BSF grant numbers 2110397 (NSF) and 2020704 (BSF).

© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 15–35, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_3](https://doi.org/10.1007/978-3-031-10769-6_3)

this amounts to providing proofs of unsatisfiability. The main challenges are justifying a combination of theory-specific algorithms while keeping the solver performant and providing enough details to allow *scalable* proof checking, i.e., checking that is fundamentally simpler than solving. Moreover, while proof production is well understood for propositional reasoning and common theories, that is not the case for more expressive theories, such as the theory of strings, or for more advanced solver operations such as formula preprocessing.

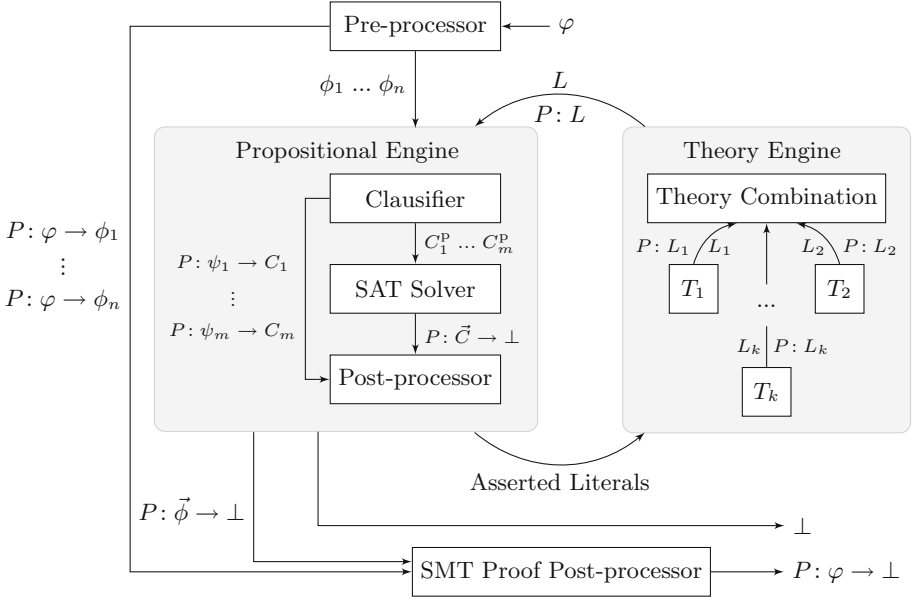
We present a new, flexible proof-production architecture for versatile, industrial-strength SMT solvers and discuss its integration into the `cvc5` solver [5]. The architecture (Sect. 2) aims to facilitate the implementation effort via modular proof production and internal proof checking, so that more critical components can be enabled when generating proofs. We provide some details on the core proof calculus and how proofs are produced (Sect. 3), in particular how we support eager and lazy proof production with built-in proof reconstruction (Sect. 3.2). This feature is particularly important for substitution and rewriting techniques, facilitating the instrumentation of notoriously challenging functionalities, such as simplification under global assumptions [6, Section 6.1] and string solving [40, 46, 48], to produce detailed proofs. Finally, we describe (Sect. 5) how the architecture is leveraged to produce detailed proofs for most of the theory reasoning, critical preprocessing, and underlying SAT solving of `cvc5`. We evaluate proof production in `cvc5` (Sect. 6) by measuring the proof overhead and the proof quality over an extensive set of benchmarks from SMT-LIB [8].

In summary, *our contributions* are a flexible proof-producing architecture for state-of-the-art SMT solvers, its implementation in `cvc5`, the production of detailed proofs for simplification under global assumptions and the full theory of strings, and initial experimental evidence that proof-production overhead is acceptable and detailed proofs can be generated for a majority of the problems.

**Preliminaries.** We assume the usual notions and terminology of many-sorted first-order logic with equality ( $\approx$ ) [29]. We consider signatures  $\Sigma$  all containing the distinguished Boolean sort `Bool`. We adopt the usual definitions of well-sorted  $\Sigma$ -terms, with literals and formulas as terms of sort `Bool`, and  $\Sigma$ -interpretations. A  $\Sigma$ -theory is a pair  $T = (\Sigma, \mathbf{I})$  where  $\mathbf{I}$ , the *models* of  $T$ , is a class of  $\Sigma$ -interpretations closed under variable reassignment. A  $\Sigma$ -formula  $\varphi$  is *T-valid* (resp., *T-unsatisfiable*) if it is satisfied by all (resp., no) interpretations in  $\mathbf{I}$ . Two  $\Sigma$ -terms  $s$  and  $t$  of the same sort are *T-equivalent* if  $s \approx t$  is *T-valid*. We write  $\vec{a}$  to denote a tuple  $(a_1, \dots, a_n)$  of elements, with  $n \geq 0$ . Depending on context, we will abuse this notation and also denote the set of the tuple's elements or, in case of formulas, their conjunction. Similarly, for term tuples  $\vec{s}, \vec{t}$  of the same length and sort, we will write  $\vec{s} \approx \vec{t}$  to denote the conjunction of equalities between their respective elements.

## 2 Proof-Production Architecture

Our proof-production architecture is intertwined with the CDCL( $\mathcal{T}$ ) architecture [43], as shown in Fig. 1. Proofs are produced and stored modularly by each solving component, which also checks they meet the expected proof structure



**Fig. 1.** Flexible proof-production architecture for CDCL( $\mathcal{T}$ )-based SMT solvers. In the above,  $\psi_i \in \{\vec{\phi}, \vec{L}\}$  for each  $i$ , with  $\psi_i$  not necessarily distinct from  $\psi_{i+1}$ .

for that component, as described below. Proofs are combined only when needed, via post-processing. The *pre-processor* receives an input formula  $\varphi$  and simplifies it in a variety of ways into formulas  $\phi_1, \dots, \phi_n$ . For each  $\phi_i$ , the pre-processor stores a proof  $P: \varphi \rightarrow \phi_i$  justifying its derivation from  $\varphi$ .

The *propositional engine* receives the preprocessed formulas, and its *clausifier* converts them into a conjunctive normal form  $C_1 \wedge \dots \wedge C_l$ . A proof  $P: \psi \rightarrow C_i$  is stored for each clause  $C_i$ , where  $\psi$  is a preprocessed formula. Note that several clauses may derive from each formula. Corresponding propositional clauses  $C_1^p, \dots, C_l^p$ , where first-order atoms are abstracted as Boolean variables, are sent to the SAT solver, which checks their joint satisfiability. The propositional engine enters a loop with the *theory engine*, which considers a set of literals asserted by the SAT solver (corresponding to a model of the propositional clauses) and verifies its satisfiability modulo a *combination of theories*  $T$ . If the set is  $T$ -unsatisfiable, a lemma  $L$  is sent to the propositional engine together with its proof  $P: L$ . Note that since lemmas are  $T$ -valid, their proofs have no assumptions. The propositional engine stores these proofs and clausifies the lemmas, keeping the respective clausification proofs in the clausifier. The clausified and abstracted lemmas are sent to the SAT solver to block the current model and cause the assertion of a different set of literals, if possible. If no new set is asserted, then all the clauses  $C_1, \dots, C_m$  generated until then are jointly unsatisfiable, and the SAT solver yields a proof  $P: C_1 \wedge \dots \wedge C_m \rightarrow \perp$ . Note that the proof is in terms of the first-order clauses, as are the derivation rules that



conclude  $\perp$  from them. The propositional abstraction does not need to be represented in the proof.

The post-processor of the propositional engine connects the assumptions of the SAT solver proof with the clausifier proofs, building a proof  $P : \phi_1 \wedge \dots \wedge \phi_n \rightarrow \perp$ . Since theory lemmas are  $T$ -valid, the resulting proof only has preprocessed formulas as assumptions. The final proof is built by the SMT solver's post-processor combining this proof with the preprocessing proofs  $P : \varphi \rightarrow \phi_i$ . The resulting proof  $P : \varphi \rightarrow \perp$  justifies the  $T$ -unsatisfiability of the input formula.

### 3 The Internal Proof Calculus

In this section, we specify how proofs are represented in the internal calculus of *cvc5*. We also provide some low-level details on how proofs are constructed and managed in our implementation.

The proof rules of the internal calculus are similar to rules in other calculi for ground first-order formulas, except that they are made a little more operational by optionally having *argument* terms and *side conditions*. Each rule has the form

$$r \frac{\varphi_1 \ \dots \ \varphi_n}{\psi} \quad \text{or} \quad r \frac{\varphi_1 \ \dots \ \varphi_n \mid t_1, \dots, t_m}{\psi} \text{ if } C$$

with *identifier*  $r$ , *premises*  $\varphi_1, \dots, \varphi_n$ , *arguments*  $t_1, \dots, t_m$ , *conclusion*  $\psi$ , and *side condition*  $C$ . The argument terms are used to construct the conclusion from the premises and can be used in the side condition together with the premises.

#### 3.1 Proof Checkers and Proofs

The semantics of each proof rule  $r$  is provided operationally in terms of a *proof-rule checker* for  $r$ . This is a procedure that takes as input a list of argument terms  $\vec{t}$  and a list of premises  $\vec{\varphi}$  for  $r$ . It returns *fail* if the input is malformed, i.e., it does not match the rule's arguments and premises or does not satisfy the side condition. Otherwise, it returns a conclusion formula  $\psi$  expressing the result of applying the rule. All proof rules of the internal calculus have an associated proof-rule checker. We say that a proof rule *proves* a formula  $\psi$ , from given arguments and premises, if its checker returns  $\psi$ .

*cvc5* has an internal proof checker built modularly out of the individual proof-rule checkers. This checker is meant mostly for internal debugging during development, to help guarantee that the constructed proofs are correct. The expectation is that users will rely instead on third-party tools to check the proof certificates emitted by the solver.

A proof object is constructed internally using a data structure that we will describe abstractly here and call a *proof node*. This is a triple  $(r, \vec{N}, \vec{t})$  consisting of a rule identifier  $r$ ; a sequence  $\vec{N}$  of proof nodes, its *children*; and a sequence  $\vec{t}$  of terms, its *arguments*. The relationships between proof nodes and their children induces a directed graph over proof nodes, with edges from proofs nodes to their children. We call a single-root graph rooted at node  $N$  a *proof*. A proof  $P$  is

$$\begin{array}{lll}
\text{refl} \frac{- \mid t}{t \approx t} & \text{trans} \frac{r \approx s \quad s \approx t}{r \approx t} & \text{cong} \frac{\vec{s} \approx \vec{t} \mid f}{f(\vec{s}) \approx f(\vec{t})} \text{ if } f(\vec{s}) \text{ is well sorted} \\
\\
\text{symm} \frac{s \approx t}{t \approx s} & \text{sr} \frac{\varphi \quad \vec{\varphi} \mid \mathcal{S}, \mathcal{R}, \mathcal{D}, \psi}{\psi} \text{ if } \mathcal{S}(\varphi, \mathcal{D}(\vec{\varphi})) \uparrow \downarrow_{\mathcal{R}} = \mathcal{S}(\psi, \mathcal{D}(\vec{\varphi})) \uparrow \downarrow_{\mathcal{R}} & \\
\\
\text{eq\_res} \frac{\varphi \quad \varphi \approx \psi}{\psi} & \text{atom\_rewrite} \frac{- \mid \mathcal{R}, s}{s \approx t} \text{ if } s \downarrow_{\mathcal{R}} = t & \text{witness} \frac{- \mid k}{k \approx k \uparrow} \\
\\
\text{assume} \frac{- \mid \varphi}{\varphi} & \text{scope} \frac{\varphi \mid \varphi_1, \dots, \varphi_n}{\varphi_1 \wedge \dots \wedge \varphi_n \Rightarrow \varphi} &
\end{array}$$

**Fig. 2.** Core proof rules of the internal calculus.

*well-formed* if it is finite, acyclic, and there is a total mapping  $\Psi$  from the nodes of  $P$  to formulas such that, for each node  $N = (r, (N_1, \dots, N_m), \vec{t})$ ,  $\Psi(N)$  is the formula returned by the proof checker for rule  $r$  when given premises  $\Psi(N_1), \dots, \Psi(N_m)$  and arguments  $\vec{t}$ . For a well-formed proof  $P$  with root  $N$  and mapping  $\Psi$ , the *conclusion* of  $P$  is the formula  $\Psi(N)$ ; a *subproof* of  $P$  is any proof rooted at a descendant of  $N$  in  $P$ . For convenience, we will identify a well-formed proof with its root node from now on.

### 3.2 Core Proof Rules

In total, the internal calculus of *cvc5* consists of 155 proof rules,<sup>1</sup> which cover all reasoning performed by the SMT solver, including theory-specific rules, rules for Boolean reasoning, and others. In the remainder of this section, we describe the *core* rules of the internal calculus, which are used throughout the system, and are illustrated in Fig. 2.

**Proof Rules for Equality.** Many theory solvers in *cvc5* perform theory-specific reasoning on top of basic equational reasoning. The latter is captured by the proof rules *eq\_res*, *refl*, *symm*, *trans*, and *cong*. The first rule is used to prove a formula  $\psi$  from a formula  $\varphi$  that was proved equivalent to  $\psi$ . The rest are the standard rules for computing the congruence closure of a set of term equalities.

**Proof Rules for Rewriting, Substitution and Witness Forms.** A single *coarse-grained* rule, *sr*, is used for tracking justifications for core utilities in the SMT solver such as *rewriting* and *substitution*. This rule, together with other non-core rules with side conditions (omitted for brevity), allows the generation of coarse-grained proofs that trust the correctness of complex side conditions. Those conditions involve rewriting and substitution operations performed by *cvc5* during solving. More fine-grained proofs can be constructed from coarse-grained ones by justifying the various rewriting and substitution steps in terms of simpler proof rules. This is done with the aid of the equality rules mentioned above and the additional core rules *atom\_rewrite* and *witness*. To describe *atom\_rewrite*, *witness*, and *sr*, we first need to introduce some definitions and notations.

<sup>1</sup> See [https://cvc5.github.io/docs/cvc5-1.0.0/proofs/proof\\_rules.html](https://cvc5.github.io/docs/cvc5-1.0.0/proofs/proof_rules.html).

A *rewriter*  $\mathcal{R}$  is a function over terms that preserves equivalence in the background theory  $T$ , i.e., returns a term  $t \downarrow_{\mathcal{R}}$   $T$ -equivalent to its input  $t$ . We call  $t \downarrow_{\mathcal{R}}$  the *rewritten* form of  $t$  with respect to  $\mathcal{R}$ . Currently, cvc5 uses a handful of specialized rewriters for various purposes, such as evaluating constant terms, preprocessing input formulas, and normalizing terms during solving. Each individual rewrite step executed by a rewriter  $\mathcal{R}$  is justified in fine-grained proofs by an application of the rule `atom_rewrite`, which takes as argument both (an identifier for)  $\mathcal{R}$  and the term  $s$  the rewrite was applied to. Note that the rule's soundness requires that the rewrite step be equivalence preserving.

A (*term*) *substitution*  $\sigma$  is a finite sequence  $(t_1 \mapsto s_1, \dots, t_n \mapsto s_n)$  of oriented pairs of terms of the same sort. A *substitution method*  $\mathcal{S}$  is a function that takes a term  $r$  and a substitution  $\sigma$  and returns a new term that is the result of *applying*  $\sigma$  to  $r$ , according to some strategy. We write  $\mathcal{S}(r, \sigma)$  to denote the resulting term. We distinguish three kinds of substitution methods for  $\sigma$ : *simultaneous*, which returns the term obtained by simultaneously replacing every occurrence of term  $t_i$  in  $r$  with  $s_i$ , for  $i = 1, \dots, n$ ; *sequential*, which splits  $\sigma$  into  $n$  substitutions  $(t_1 \mapsto s_1), \dots, (t_n \mapsto s_n)$  and applies them in sequence to  $r$  using the simultaneous strategy above; and *fixed-point*, which, starting with  $r$ , repeatedly applies  $\sigma$  with the simultaneous strategy until no further subterm replacements are possible. For example, consider the application  $\mathcal{S}(y, (x \mapsto u, y \mapsto f(z), z \mapsto g(x)))$ . The steps the substitution method takes in computing its result are the following:  $y \rightsquigarrow f(z)$  if  $\mathcal{S}$  is simultaneous;  $y \rightsquigarrow f(z) \rightsquigarrow f(g(x))$  if  $\mathcal{S}$  is sequential;  $y \rightsquigarrow f(z) \rightsquigarrow f(g(x)) \rightsquigarrow f(g(u))$  if  $\mathcal{S}$  is fixed-point.

In cvc5, we use a *substitution derivation method*  $\mathcal{D}$  to derive a *contextual* substitution  $(t_1 \mapsto s_1, \dots, t_n \mapsto s_n)$  from a collection  $\vec{\varphi}$  of derived formulas. The substitution essentially orients a selection of term equalities  $t_i \approx s_i$  entailed by  $\vec{\varphi}$  and, as such, can be applied soundly to formulas derived from  $\vec{\varphi}$ .<sup>2</sup> We write  $\mathcal{D}(\vec{\varphi})$  to denote the substitution computed by  $\mathcal{D}$  from  $\vec{\varphi}$ .

Finally, cvc5 often introduces fresh variables, or *Skolem* variables, which are implicitly globally existentially quantified. This happens as a consequence of Skolemization of existential variables, lifting of if-then-else terms, and some kinds of flattening. Each Skolem variable  $k$  is associated with a term  $k \uparrow$  of the same sort containing no Skolem variables, called its *witness term*. This global map from Skolem variables to their witness term allows cvc5 to detect when two Skolem variables can be equated, as a consequence of their respective witness terms becoming equivalent in the current context [47]. Witness terms can also be used to eliminate Skolem variables at proof output time. We write  $t \uparrow$  to denote the *witness form* of term  $t$ , which is obtained by replacing every Skolem variable in  $t$  by its witness term. For example, if  $k_1$  and  $k_2$  are Skolem variables with associated witness terms  $\text{ite}(x \approx z, y, z)$  and  $y - z$ , respectively, and  $\varphi$  is the formula  $\text{ite}(x \approx k_2, k_1 \approx y, k_1 \approx z)$ , the witness form  $\varphi \uparrow$  of  $\varphi$  is the formula  $\text{ite}(x \approx y - z, \text{ite}(x \approx z, y, z) \approx y, \text{ite}(x \approx z, y, z) \approx z)$ . When a Skolem variable  $k$

<sup>2</sup> Observe that substitutions are generated dynamically from the formulas being processed, whereas rewrite rules are hard-coded in cvc5's rewriters.

appears in a proof, the **witness** proof rule is used to explicitly constrain its value to be the same as that of the term  $k\uparrow$  it abstracts.<sup>3</sup>

We can now explain the **sr** proof rule, which is parameterized by a substitution method  $\mathcal{S}$ , a rewriter  $\mathcal{R}$ , and substitution derivation method  $\mathcal{D}$ . The rule is used to transform the proof of a formula  $\varphi$  into one of a formula  $\psi$  provided that the two formulas are equal up to rewriting under a substitution derived from the premises  $\vec{\varphi}$ . Note that this rule is quite general because its conclusion  $\psi$ , which is provided as an argument, can be any formula that satisfies the side condition.

**Proof Rules for Scoped Reasoning.** Two of the core proof rules, **assume** and **scope**, enable local reasoning. Together they achieve the effect of the  $\Rightarrow$ -introduction rule of Natural Deduction. However, separating the local assumption functionality in **assume** provides more flexibility. That rule has no premises and introduces a local assumption  $\varphi$  provided as an argument. The **scope** rule is used to *close the scope* of the local assumptions  $\varphi_1, \dots, \varphi_n$  made to prove a formula  $\varphi$ , inferring the formula  $\varphi_1 \wedge \dots \wedge \varphi_n \Rightarrow \varphi$ .

We say that  $\varphi$  is a *free assumption* in proof  $P$  if  $P$  has a node (**assume**,  $()$ ,  $\varphi$ ) that is not a subproof of a **scope** node with  $\varphi$  as one of its arguments. A proof is *closed* if it has no free assumptions, and *open* otherwise.

**Soundness.** All proof rules other than **assume** are *sound* with respect to the background theory  $T$  in the following sense: if a rule proves a formula  $\psi$  from premises  $\vec{\varphi}$ , every model of  $T$  that satisfies  $\vec{\varphi}$ , and assigns the same values to Skolem variables and their respective witness term, satisfies  $\psi$  as well. Based on this and a simple structural induction argument, one can show that well-formed closed proofs have  $T$ -valid conclusions. In contrast, open proofs have conclusions that are  $T$ -valid only under assumptions. More precisely, in general, if  $\vec{\varphi}$  are all the free assumptions of a well-formed proof  $P$  with conclusion  $\psi$  and  $\vec{k}$  are all the Skolem variables introduced in  $P$ , then  $\vec{k} \approx \vec{k}\uparrow \wedge \vec{\varphi} \Rightarrow \psi$  is  $T$ -valid.

### 3.3 Constructing Proof Nodes

We have implemented a library of *proof generators* that encapsulates common patterns for constructing proof nodes. We assume a method `getProof` that takes the proof generator  $g$  and a formula  $\varphi$  as input and returns a proof node with conclusion  $\varphi$  based on the information in  $g$ . During solving, `cvc5` uses a combination of *eager* and *lazy* proof generation. In general terms, eager proof generation involves constructing proof nodes for inference steps at the time those steps are taken during solving. Eager proof generation may be required if the computation state pertinent to that inference cannot be easily recovered later. In contrast, lazy proof generation occurs for inferred formulas associated with proof generators that can do internal bookkeeping to be able to construct proof nodes for the formula *after* solving is completed. Depending on the formula, different kinds of proof generators are used. For brevity, we only describe in detail (see Sect. 3.2)

<sup>3</sup> The proof rules that account for the introduction of Skolem variables in the first place are not part of the core set and so are not discussed here.

---

**Algorithm 1.** Proof generation for term-conversion generators, rewrite-once policy.  $B$  is a lazy proof builder,  $R$  a map from terms to their converted form, and  $c_{\text{pre}}, c_{\text{post}}$  are sets of pairs of equalities and the proof generators justifying them.

---

**getProof**( $g, \varphi$ ) where  $g$  contains  $c_{\text{pre}}, c_{\text{post}}$  and  $\varphi$  is  $t_1 \approx t_2$

```

1:  $B := \emptyset, R := \emptyset$ 
2: getTermConv( $t_1, c_{\text{pre}}, c_{\text{post}}, B, R$ )
3: if  $R[t_1] \neq t_2$  then fail else return getProof( $B, t_1 \approx R[t_1]$ )

getTermConv( $s, c_{\text{pre}}, c_{\text{post}}, B, R$ ), where  $s = f(s_1, \dots, s_n)$ 
1: if  $s$  in  $\text{dom}(R)$  then return
2: if  $(s \approx s', g') \in c_{\text{pre}}$  for some  $s', g'$  then
3:    $R[s] := s', \text{addLazyStep}(B, s \approx s', g')$ 
4:   return
5: for  $1 \leq i \leq n$  do getTermConv( $s_i, c_{\text{pre}}, c_{\text{post}}, B, R$ )
6:  $R[s] := r$ , where  $r = f(R[s_1], \dots, R[s_n])$ 
7: if  $s \neq r$  then addStep( $B, \text{cong}, (s_1 \approx R[s_1], \dots, s_n \approx R[s_n]), f$ )
8: else addStep( $B, \text{rfl}, (), s \approx s$ )
9: if  $(r \approx r', g') \in c_{\text{post}}$  for some  $r', g'$  then
10:   $R[s] := r', \text{addLazyStep}(B, r \approx r', g'), \text{addStep}(B, \text{trans}, (s \approx r, r \approx r'), ())$ 

```

---

the proof generator most relevant to the core calculus, the *term-conversion proof generator*, targeted for substitution and rewriting proofs.

## 4 Proof Reconstruction for Substitution and Rewriting

Once it determines that the input formulas  $\varphi_1, \dots, \varphi_n$  are jointly unsatisfiable, the SMT solver has a reference to a proof node  $P$  that concludes  $\perp$  from the free assumptions  $\varphi_1, \dots, \varphi_n$ . After the post-processor is run, the (closed) proof ( $\text{scope}, P', (\varphi_1, \dots, \varphi_n)$ ) is then generated as the final proof for the user, where  $P'$  is the result of optionally expanding coarse-grained steps (in particular, applications of the rule *sr*) in  $P$  into fine-grained ones. To do so, we require the following algorithm for generating *term-conversion* proofs.

In particular, we focus on equalities  $t \approx s$  whose proof can be justified by a set of steps that replace subterms of  $t$  until it is syntactically equal to  $s$ . We assume these steps are provided to a *term-conversion proof generator*. Formally, a term-conversion proof generator  $g$  is a pair of sets  $c_{\text{pre}}$  and  $c_{\text{post}}$ . The set  $c_{\text{pre}}$  (resp.,  $c_{\text{post}}$ ) contains pairs of the form  $(t \approx s, g_{t,s})$  indicating that  $t$  should be replaced by  $s$  in a preorder (resp., postorder) traversal of the terms that  $g$  processes, where  $g_{t,s}$  is a proof generator that can prove the equality  $t \approx s$ . We require that neither  $c_{\text{pre}}$  nor  $c_{\text{post}}$  contain multiple entries of the form  $(t \approx s_1, g_1)$  and  $(t \approx s_2, g_2)$  for distinct  $(s_1, g_1)$  and  $(s_2, g_2)$ .

The procedure for generating proofs from a term-conversion proof generator  $g$  is given in Algorithm 1. When asked to prove an equality  $t_1 \approx t_2$ , **getProof** traverses the structure of  $t_1$  and applies steps from the sets  $c_{\text{pre}}$  and  $c_{\text{post}}$  from  $g$ .

The traversal is performed by the auxiliary procedure `getTermConv` which relies on two data structures. The first is a *lazy proof builder*  $B$  that stores the intermediate steps in the overall proof of  $t_1 \approx t_2$ . The proof builder is given these steps either via `addStep`, as a concrete triple with the proof rule, a list of premise formulas, and a list of argument terms, or as a *lazy* step via `addLazyStep`, with a formula and a reference to another generator that can prove that formula. The second data structure is a mapping  $R$  from terms to terms that is updated (using array syntax in the pseudo-code) as the converted form of terms is computed by `getTermConv`. For any term  $s$ , executing `getTermConv(s, cpre, cpost, B, R)` will result in  $R[s]$  containing the converted form of  $s$  according to the rewrites in  $c_{\text{pre}}$  and  $c_{\text{post}}$ , and  $B$  storing a proof step for  $s \approx R[s]$ . Thus, the procedure `getProof` succeeds when, after invoking `getTermConv(t1, cpre, cpost, B, R)` with  $B$  and  $R$  initially empty, the mapping  $R$  contains  $t_2$  as the converted form of  $t_1$ . The proof for the equality  $t_1 \approx R[t_1]$  can then be constructed by calling `getProof` on the lazy proof builder  $B$ , based on the (lazy) steps stored in it.

Each subterm  $s$  of  $t_1$  is traversed only once by `getTermConv` by checking whether  $R$  already contains the converted form of  $s$ . When that is not the case,  $s$  is first preorder processed. If  $c_{\text{pre}}$  contains an entry indicating that  $s$  rewrites to  $s'$ , this rewrite step is added to the lazy proof builder and the converted form  $R[s]$  of  $s$  is set to  $s'$ . Otherwise, the immediate subterms of  $s$ , if any, are traversed and then  $s$  is postorder processed. The converted form of  $s$  is set to some term  $r$  of the form  $f(R[s_1], \dots, R[s_n])$ , considering how its immediate subterms were converted. Note that  $B$  will contain steps for  $\vec{s} \approx R[\vec{s}]$ . Thus, the equality  $s \approx r$  can be proven by congruence for function  $f$  with these premises if  $s \neq r$ , and by reflexivity otherwise. Furthermore, if  $c_{\text{post}}$  indicates that  $r$  rewrites to  $r'$ , then this step is added to the lazy proof builder; a transitivity step is added to prove  $s \approx r'$  from  $t \approx r$  and  $r \approx r'$ ; and the converted form  $R[s]$  is set to  $r'$ .

*Example 1.* Consider the equality  $t \approx \perp$ , where  $t = f(b) + f(a) < f(a - 0) + f(b)$ , and suppose the conversion of  $t$  is justified by a term-conversion proof generator  $g$  containing the sets  $c_{\text{pre}} = \{(f(b) + f(a) \approx f(a) + f(b), g^{\text{AC}}), (a - 0 \approx a, g_0^{\text{Arith}})\}$  and  $c_{\text{post}} = \{(f(a) + f(b) < f(a) + f(b) \approx \perp, g_1^{\text{Arith}})\}$ . The generator  $g^{\text{AC}}$  provides a proof based on associative and commutative reasoning, whereas  $g_0^{\text{Arith}}$  and  $g_1^{\text{Arith}}$  provide proofs based on arithmetic reasoning. Invoking `getProof(g, t  $\approx$   $\perp$ )` initiates the traversal with `getTermConv(t, cpre, cpost,  $\emptyset$ ,  $\emptyset$ )`. Since  $t$  is not in the conversion map, it is preorder processed. However, as it does not occur in  $c_{\text{pre}}$ , nothing is done and its subterms are traversed. The subterm  $f(b) + f(a)$  is equated to  $f(a) + f(b)$  in  $c_{\text{pre}}$ , justified by  $g^{\text{AC}}$ . Therefore  $R$  is updated with  $R[f(b) + f(a)] = f(a) + f(b)$  and the respective lazy step is added to  $B$ . The subterms of  $f(b) + f(a)$  are not traversed, therefore the next term to be traversed is  $f(a - 0) + f(b)$ . Since it does not occur in  $c_{\text{pre}}$ , its subterm  $f(a - 0)$  is traversed, which analogously leads to the traversal of  $a - 0$ . As  $a - 0$  does occur in  $c_{\text{pre}}$ , both  $R$  and  $B$  are updated accordingly and the processing of its parent  $f(a - 0)$  resumes. A congruence step added to  $B$  justifies its conversion to  $f(a)$  being added to  $R$ .

No more additions happen since  $f(a)$  does not occur in  $c_{\text{post}}$ . Analogously,  $R$  and  $B$  are updated with  $f(b)$  not changing and  $f(a - 0) + f(b)$  being converted into  $f(a) + f(b)$ . Finally, the processing returns to the initial term  $t$ , which has been converted to  $R[f(b) + f(a)] < R[f(a + 0) + f(b)]$ , i.e.,  $f(a) + f(b) < f(a) + f(b)$ . Since this term is equated to  $\perp$  in  $c_{\text{post}}$ , justified by  $g_1^{\text{Arith}}$ , the respective lazy step is added to  $B$ , as well as a transitivity step to connect  $f(b) + f(a) < f(a - 0) + f(b) \approx f(a) + f(b) < f(a) + f(b)$  and  $f(a) + f(b) < f(a) + f(b) \approx \perp$ . At this point, the execution terminates with  $R[f(b) + f(a) < f(a + 0) + f(b)] = \perp$ , as expected. A proof for  $t \approx \perp$  with the following structure can then be extracted from  $B$ :

$$\begin{array}{c}
P_0 : \text{cong} \frac{\text{Lazy} \frac{g^{\text{AC}}}{f(b) + f(a) \approx f(a) + f(b)} \quad P_1 \mid <}{f(b) + f(a) < f(a - 0) + f(b) \approx f(a) + f(b) < f(a) + f(b)} \quad P_2 : \text{refl} \frac{- \mid f(b) \approx f(b)}{f(b) \approx f(b)} \\
\\
\text{trans} \frac{P_0 \quad \text{Lazy} \frac{g_1^{\text{Arith}}}{f(a) + f(b) < f(a) + f(b) \approx \perp}}{f(b) + f(a) < f(a - 0) + f(b) \approx \perp} \quad P_1 : \text{cong} \frac{\text{Lazy} \frac{g_0^{\text{Arith}}}{a - 0 \approx a} \quad f}{f(a - 0) \approx f(a)} \quad P_2 \mid +
\end{array}$$

We use several extensions to the procedures in Algorithm 1. Notice that this procedure follows the policy that terms on the right-hand side of conversion steps (equalities from  $c_{\text{pre}}$  and  $c_{\text{post}}$ ) are not traversed further. The procedure `getTermConv` is used by term-conversion proof generators that have the *rewrite-once* policy. A similar procedure which additionally traverses those terms is used by term-conversion proof generators that have a *rewrite-to-fixpoint* policy.

We now show how the term-conversion proof generator can be used for reconstructing fine-grained proofs from coarse-grained ones. In particular we focus on proofs  $P_{\psi_1}$  of the form  $(\text{sr}, (Q_{\psi_0}, \vec{Q}), (\mathcal{S}, \mathcal{R}, \mathcal{D}, \psi))$ . Recall from Fig. 2 that the proof rule `sr` concludes a formula  $\psi$  that can be shown equivalent to the formula  $\psi_0$  proven by  $Q_{\psi_0}$  based on a substitution derived from the conclusions of the nodes  $\vec{Q}$ . A proof like  $P_{\psi_1}$  above can be transformed to one that involves (atomic) theory rewrites and equality rules only. We show this transformation in two phases. In the first phase, the proof is expanded to:

$$(\text{eq\_res}, (Q_{\psi_0}, (\text{trans}, (R_0, (\text{symm}, R_1))))))$$

with  $R_i = (\text{trans}, ((\text{subs}, \vec{Q}_{\vec{\varphi}}, (\mathcal{S}, \mathcal{D}, \psi_i)), (\text{rewrite}, (), (\mathcal{R}, \mathcal{S}(\psi_i, \mathcal{D}(\vec{\varphi}))))))$  for  $i \in \{0, 1\}$  where  $\vec{\varphi}$  are the conclusions of  $\vec{Q}_{\vec{\varphi}}$ , and `subs` and `rewrite` are auxiliary proof rules used for further expansion in the second phase. We describe them next.

*Substitution Steps.* Let  $P_{t \approx s}$  be the subproof  $(\text{subs}, \vec{Q}_{\vec{\varphi}}, (\mathcal{S}, \mathcal{D}, t))$  of  $R_i$  above proving  $t \approx s$  with  $s = \mathcal{S}(\psi_i, \mathcal{D}(\vec{\varphi}))$  and  $\mathcal{D}(\vec{\varphi}) = (t_1 \mapsto s_1, \dots, t_n \mapsto s_n)$ . Substitution steps can be expanded to fine-grained proofs using a term-conversion proof generator. First, for each  $j = 1, \dots, n$ , we construct a proof of  $t_j \approx s_j$ , which involves simple transformations on the proofs of  $\vec{\varphi}$ . Suppose we store all of these in an eager proof generator  $g$ . If  $\mathcal{S}$  is a simultaneous or fixed-point substitution, we then build a single term-conversion proof generator  $C$ , which

recall is modeled as a pair of mappings  $(c_{\text{pre}}, c_{\text{post}})$ . We add  $(t_j \approx s_j, g)$  to  $c_{\text{pre}}$  for all  $j$ . We use the rewrite-once policy for  $C$  if  $\mathcal{S}$  is a simultaneous substitution, and the rewrite-fixed-point policy for  $C$  otherwise. We then replace the proof  $P_{t \approx s}$  by  $\text{getProof}(C, t \approx s)$ , which runs the procedure in Algorithm 1. Otherwise, if  $\mathcal{S}$  is a sequential substitution, we construct a term-conversion generator  $C_j$  for *each*  $j$ , initializing it so that its  $c_{\text{pre}}$  set contains the single rewrite step  $(t_j \approx s_j, g)$  and uses a rewrite-once policy. We then replace the proof  $P_{t \approx s}$  by  $(\text{trans}, (P_1, \dots, P_n))$  where, for  $j = 1, \dots, n$ :  $P_j$  is generated by  $\text{getProof}(C_j, s_{j-1} \approx s_j)$ ;  $s_0 = t$ ;  $s_i$  is the result of the substitution  $\mathcal{D}(\vec{\varphi})$  after the first  $i$  steps; and  $s_n = s$ .

*Rewrite Steps.* Let  $P$  be the proof node  $(\text{rewrite}, (), (\mathcal{R}, t))$ , which proves the equality  $t \approx t \uparrow \downarrow_{\mathcal{R}}$ . During reconstruction, we replace  $P$  with a proof involving only fine-grained rules, depending on the rewrite method  $\mathcal{R}$ . For example, if  $\mathcal{R}$  is the core rewriter, we run the rewriter again on  $t$  in proof tracking mode. Normally, the core rewriter performs a term traversal and applies atomic rewrites to completion. In proof tracking mode, it also return two lists, for pre- and post-rewrites, of steps  $(t_1 \approx s_1, g), \dots, (t_n \approx s_n, g)$  where  $g$  is a proof generator that returns  $(\text{atom.rewrite}, (), (\mathcal{R}, t_i))$  for all equalities  $t_i \approx s_i$ . Furthermore, for each Skolem  $k$  that is a subterm of  $t$ , we construct the rewrite steps  $(k \approx k \uparrow, g')$  where  $g'$  is a proof generator that returns  $(\text{witness}, (), (k))$  for equalities  $k \approx k \uparrow$ . We add these rewrite proof steps to a term-conversion generator  $C$  with rewrite-fixed-point policy, and replace  $P$  by  $\text{getProof}(C, t \approx t \uparrow \downarrow_{\mathcal{R}})$ .

## 5 SMT Proofs

Here we briefly describe each component shown in Sect. 2 and how it produces proofs with the infrastructure from Sects. 3 and 3.2.

### 5.1 Preprocessing Proofs

The *pre-processor* transforms an input formula  $\varphi$  into a list of formulas to be given to the core solver. It applies a sequence of *preprocessing passes*. A pass may *replace* a formula  $\varphi_i$  with another one  $\phi_i$ , in which case it is responsible for providing a proof of  $\varphi_i \approx \phi_i$ . It may also append a new formula  $\phi$  to the list, in which case it is responsible for providing a proof for it. We use a (lazy) proof generator that tracks these proofs, maintaining the invariant that a proof can be provided for all (preprocessed) formulas when requested. We have instrumented proof production for the most common preprocessing passes, relying heavily on the *sr* rule to model transformations such as expansion of function definitions and, with witness forms, Skolemization and if-then-else elimination [6].

*Simplification Under Global Assumptions.* *cvc5* aggressively learns literals that hold globally by performing Boolean constraint propagation over the input formula. When a learned literal corresponds to a variable elimination (e.g.,  $x \approx 5$  corresponds to  $x \mapsto 5$ ) or a constant propagation (e.g.,  $P(x)$  corresponds to



$P(x) \mapsto \top$ ), we apply the corresponding (term) substitution to the input. This application is justified via **sr**, while the derivation of the globally learned literals is justified via clausification and resolution proofs, as explained in Sect. 5.3.

The key features of our architecture that make it feasible to produce proofs for this simplification are the automatic reconstruction of **sr** steps and the ability to customize the strategy for substitution application during reconstruction, as detailed in Sect. 3.2. When a new variable elimination  $x \mapsto t$  is learned, old ones need to be normalized to eliminate any occurrences of  $x$  in their right-hand sides. Computing the appropriate simultaneous substitution for all eliminations requires quadratically many traversals over those terms. We have observed that the size of substitutions generated by this preprocessing pass can be very large (with thousands of entries), which makes this computation prohibitively expensive. Using the fixed-point strategy, however, the reconstruction for the **sr** steps can apply the substitution efficiently and its complexity depends on how many applications are necessary to reach a fix-point, which is often low in practice.

## 5.2 Theory Proofs

The theory engine produces lemmas, as disjunctions of literals, from an individual theory or a combination of them. In the first case, the lemma’s proof is provided directly by the corresponding theory solver. In the second case, a theory solver may produce a lemma  $\psi$  containing a literal  $\ell$  derived by some other theory solver from literals  $\bar{\ell}$ . A lemma over the combined theory is generated by replacing  $\ell$  in  $\psi$  by  $\bar{\ell}$ . This regression process, which is similar to the computation of *explanations* during solving, is repeated until the lemma contains only input literals. The proof of the final lemma then uses rules like **sr** to combine the proofs of the intermediate literals derived locally in various theories and their replacement by input literals in the final lemma.

*Equality and Uninterpreted Function (EUF) Proofs.* The EUF solver can be easily instrumented to produce proofs [31, 42] with equality rules (see Fig. 2). In *cvc5*, term equivalences are also derived via rewriting in some other theory  $T$ : when a function from  $T$  has all of its arguments inferred to be congruent to  $T$ -values, it may be rewritten into a  $T$ -value itself, and this equivalence asserted. Such equivalences are justified via **sr** steps. Since generating equality proofs incurs minimal overhead [42] and rewriting proofs are reconstructed lazily, EUF proofs are generated during solving and stored in an eager proof generator.

*Extensional Arrays and Datatypes Proofs.* While these two theories differ significantly, they both combine equality reasoning with rules for handling their particular operators. For arrays, these are rules for **select**, **store**, and array extensionality (see [36, Sec. 5]). For datatypes, they are rules reflecting the properties of *constructors* and *selectors*, as well as acyclicity. The justifications for lemmas are also generated eagerly and stored in an eager proof generator.

*Bit-Vector Proofs.* The bit-vector solver applies bit-blasting to reduce bit-vector problems to equisatisfiable propositional problems. Thus, its lemmas amount

to the rewriting of the bit-vector literals into Boolean formulas, which will be solved and proved by the propositional engine. The bit-vector lemmas are proven lazily, analogous to `sr` steps, with the difference that the reconstruction uses the bit-blaster in the bit-vector solver instead of the rewriter.

*Arithmetic Proofs.* The *linear* arithmetic solver is based on the simplex algorithm [24], and each of its lemmas is the negation of an unsatisfiable conjunction of inequalities. Farkas’ lemma [30, 49] guarantees that there exists a linear combination of these inequalities equivalent to  $\perp$ . The coefficients of the combination are computed during solving with minimal overhead [38], and the equivalence is proven with an `sr` step. To allow the rewriter to prove this equivalence, the bounds of the inequalities are scaled by constants and summed during reconstruction. Integer reasoning is proved through rules for branching and integer bound tightening, recorded eagerly.

*Non-linear* arithmetic lemmas are generated from incremental linearization [16] or cylindrical algebraic coverings [1]. The former can be proven via propositional and basic arithmetic rules, with only a few, such as the tangent plane lemma, needing a dedicated proof rule. The latter requires two complex rules that are not inherently simpler than solving, albeit not as complex as those for regular CAD-based theory solvers [2]. We point out that checking these rules would require a significant portion of CAD-related theory, whose proper formalization is still an open, if actively researched, problem [18, 25, 34, 41, 53].

*Quantifier Proofs.* Quantified formulas not Skolemized during pre-processing are handled via instantiation, which produces theory lemmas of the form  $(\forall \vec{x} \varphi) \Rightarrow \varphi\sigma$ , where  $\sigma$  is a grounding substitution. An instantiation rule proves them independently of how the substitution was actually derived, since any well-typed one suffices for soundness.

*String Proofs.* The strings solver applies a layered approach, distinguishing between core [40] and extended operators [48]. The core operators consist of (dis)equalities between string concatenations and length constraints. Reasoning over them is proved by a combination of equality and linear integer arithmetic proofs, as well as specific string rules. The extended operators are reduced to core ones via formulas with bounded quantifiers. The reductions are proven with rules defining each extended function’s semantics, and `sr` steps justifying the reductions. Finally, regular membership constraints are handled by string rules that unfold occurrences of the Kleene star operator and split up regular expression concatenations into different parts. Overall, the proofs for the strings theory solver encompass not only string-specific reasoning but also equality, linear integer arithmetic, and quantifier reasoning, as well as substitution and rewriting.

*Unsupported.* The theory solvers for the theories of floating-point arithmetic, sequences, sets and relations, and separation logic are currently not proof-producing in `cvc5`. These are relatively new or non-standard theories in SMT and have not been our focus, but we intend to produce proofs for them in the future.

**Table 1.** Cumulative solving times (s) on benchmarks solved by all configurations, with the slowdown versus CVC+S in parentheses.

Logics	#	CVC+OS	CVC+S	CVC+SP	CVC+SPR
NON-BVs	116,321	164k	166k	284k (1.7×)	299k (1.8×)
BVs	29,192	45k	57k	150k (2.6×)	224k (3.9×)

### 5.3 Propositional Proofs

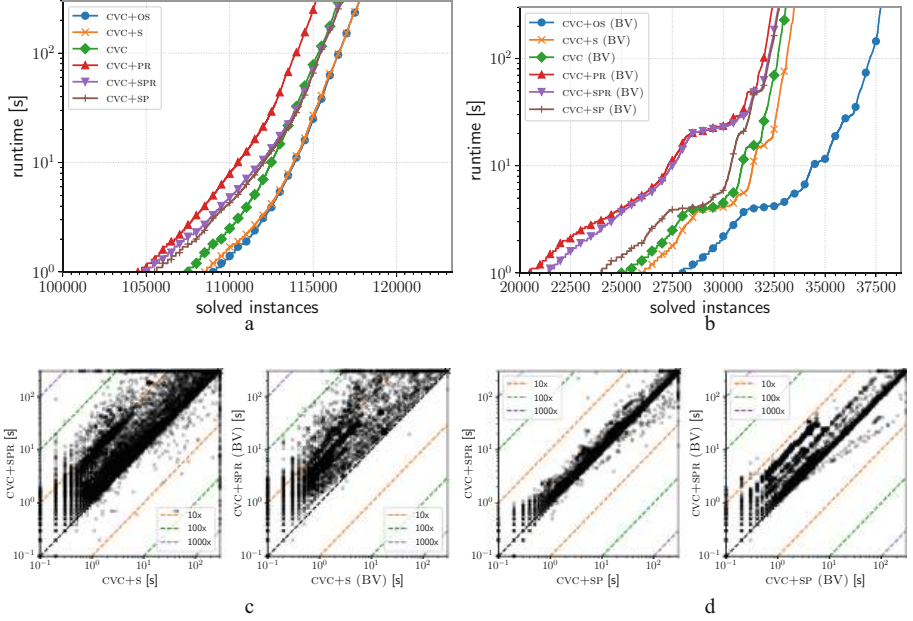
Propositional proofs justify both the conversion of preprocessed input formulas and theory lemmas into conjunctive normal form (CNF) and the derivation of  $\perp$  from the resulting clauses. CNF proofs are a combination of Boolean transformations and introductions of Boolean formulas representing the definition of Tseytin variables, used to ensure that the CNF conversion is polynomial. The clausifier uses a lazy proof builder which stores the clausification steps eagerly, with the preprocessed input formulas as assumptions, and the theory lemmas as lazy steps, with associated proof generators.

For Boolean reasoning, *cvc5* uses a version of MiniSat [27] instrumented to produce resolution proofs. It uses a lazy proof builder to record resolution steps for learned clauses as they are derived (see [7, Chap 1] for more details) and to lazily build a refutation with only the resolution steps necessary for deriving  $\perp$ . The resolution rule, however, is ground first-order resolution, since the proofs are in terms of the first-order clauses rather than their propositional abstractions.

## 6 Evaluation

In this section, we discuss an initial evaluation of our implementation in *cvc5* of the proof-production architecture presented in this paper. In the following, we denote different configurations of *cvc5* by CVC plus some suffixes. A configuration using variable and clause elimination in the SAT solver [26], symmetry breaking [23] in the EUF solver, and black-box SAT solving in the bit-vector (BV) solver, is denoted by the suffix O. These techniques are currently incompatible with the proof production architecture. Other *cvc5* techniques for which we do not yet support fine-grained proofs, however, are active and have their inferences registered in the proofs as trusted steps. A configuration that includes simplification under global assumptions is denoted by S; one that includes producing proofs by P; and one that additionally reconstructs proofs by R. The default configuration of *cvc5* is CVC+OS.

We split our evaluation into measuring the proof-production cost as well as the performance impact of making key techniques proof-producing; the proof reconstruction overhead; and the coverage of the proof production. We also comment on how *cvc5*’s proofs compare with CVC4’s proofs. Note that the internal proof checking described in Sect. 3, which was invaluable for a correct implementation, is disabled for evaluating performance. Experiments ran on a cluster with



**Fig. 3.** (a) Cactus plot for NON-BVs (b) Cactus plot for BVs (c) Scatter plot of overall proof cost (d) Reconstruction cost

Intel Xeon E5-2620 v4 CPUs, with 300s and 8GB of RAM for each solver and benchmark pair. We consider 162,060 unsatisfiable problems from SMT-LIB [8], across all logics except those with floating point arithmetic, as determined by `cvc5` [5, Sec. 4]. We split them into 38,732 problems with the BV theory (the BVs set) and 123,328 problems without (the NON-BVs set).

*Proof Production Cost.* The cost of proof production is summarized in Table 1 and Figs. 3a to 3d. The impact of running without `o` is negligible overall in NON-BVs, but steep for BVs, both in terms of solving time and number of problems solved, as evidenced by the table and Fig. 3b respectively. This is expected given the effectiveness of combining bit-blasting with black-box SAT solvers. The overhead of `P` is similar for both sets, although more pronounced in BVs. While the total time is around double that of `CVC+S`, Fig. 3c shows a finer distribution, with most problems having a less significant overhead. Moreover, the total number of problems solved is quite similar, as shown in Figs. 3a and 3b, particularly for NON-BVs. The difference in overhead due to `P` between the BVs and NON-BVs sets can be attributed to the cost of managing large proofs, which are more common in BVs. This stems from the well-known blow-up in problem size incurred by bit-blasting, which is reflected in the proofs.

The cost of generating fine-grained steps for the `sr` rule and for the similarly reconstructed theory-specific steps mentioned in Sect. 5, varies again between

the two sets, but more starkly. While for NON-BVs the overall solving time and number of problems solved are very similar between CVC+SP and CVC+SPR, for the BVs set CVC+SPR is significantly slower overall. This difference again arises mainly because of the increased proof sizes. Nevertheless, R leads to only a small increase in unsolved problems in BVs, as shown in Fig. 3b.

The importance of being able to produce proofs for simplification under global assumptions is made clear by Fig. 3a: the impact of disabling *s* is virtually the same as that of adding *P*; moreover, CVC+SPR significantly outperforms CVC+PR. In Fig. 3b the difference is less pronounced but still noticeable.

*Proofs Coverage.* When using techniques that are not yet fully proof-producing, but still active, *cvc5* inserts *trusted steps* in the proof. These are usually steps whose checking is not inherently simpler than solving. They effectively represent holes in the proof, but are still useful for users who avail themselves of powerful proof-checking techniques. Trusted steps are commonly used when integrating SMT solvers into proof assistants [11, 28, 51].

The percentage of CVC+SPR proofs *without* trusted steps is 92% for BVs and 80% for NON-BVs. That is to say, out of 145,683 proofs, 120,473 of them are fully fine-grained proofs. The vast majority of the trusted steps in the remaining proofs are due to theory-specific preprocessing passes that are not yet fully proof-producing. In NON-BVs, the occurrence of trusted steps is heavily dependent on the specific SMT-LIB logic, as expected. Common offenders are logics with datatypes, with trusted steps for acyclicity checks, and quantified logics, with trusted steps for certain  $\alpha$ -equivalence eliminations. In non-linear real arithmetic logics, all cylindrical algebraic coverings proofs are built with trusted steps (see Sect. 5.2), but we note this is the state of the art for CAD-based proofs. As for non-linear integer arithmetic logics, our proof support is still in its early stages, so a significant portion of their theory lemmas are trusted steps.

We stress the extent of our coverage for string proofs, which were previously unsupported by any SMT solver. In the string logics without length constraints, 100% of the proofs are fully fine-grained. This rate goes down to 80% in the logics with length. For the remaining 20%, the overwhelming majority of the trusted steps are for theory-specific preprocessing or some particular string or linear arithmetic inference within the proof of a theory lemma.

*Comparison with CVC4 Proofs.* We compare the proof coverage of *cvc5* versus CVC4. The *cvc5* proof production replaces CVC4’s [32, 36], which was incomplete and monolithic. CVC4 did not produce proofs at all for strings, substitutions, rewriting, preprocessing, quantifiers, datatypes, or non-linear arithmetic. In particular, simplification over global assumptions had to be disabled when producing proofs. In fragments supported by both systems, CVC4’s proofs are at most as detailed as *cvc5*’s. The only superior aspect of CVC4’s proof production was to support proofs from external SAT solvers [45] used in the BV solver, which are very significant for solving performance, as shown above. Integrating this feature into *cvc5* is left as future work, but we note that there is no limitation in the proof architecture that would prevent it. We also point out that *cvc5* produces resolution proofs for the bit-blasted BV constraints, which can

be checked in polynomial time, whereas external SAT solvers produce DRAT proofs [33] (or reconstructions of them via other tools [19, 20, 37, 39]), which can take exponential time to check. So there is a significant trade-off to be considered.

## 7 Related Work

Two significant proof-producing state-of-the-art SMT solvers are z3 [22] and veriT [14]. Both can have their proofs successfully reconstructed in proof assistants [3, 12, 13, 51]. They can produce detailed proofs for the propositional and theory reasoning in EUF and linear arithmetic, as well as for quantifiers. However, z3's proofs are coarse-grained for preprocessing and rewriting, and for bit-vector reasoning, which complicates proof checking. Moreover, to the best of our knowledge, z3 does not produce proofs for its other theories. In contrast, veriT can produce fine-grained proofs for preprocessing and rewriting [6], which has led to a better integration with Isabelle/HOL [51]. However, it does so eagerly, which requires a tight integration between the preprocessing and the proof-production code. In addition, it does not support simplification under global assumptions when producing proofs, which significantly impacts its performance. Other proof-producing SMT solvers are MathSAT5 [17] and SMTInterpol [15]. They produce resolution proofs and theory proofs for EUF, linear arithmetic, and, in SMTInterpol's case, array theories. Their proofs are tailored towards unsatisfiable core and interpolant generation, rather than external certification. Moreover, they do not seem to provide proofs for preprocessing, clausification or rewriting.

While cvc5 is possibly the only proof-producing solver for the full theory of strings, CERTISTR [35] is a certified solver for the fragment with concatenation and regular expressions. It is automatically generated from Isabelle/HOL [44] but is significantly less performant than cvc5, although a proper comparison would need to account for proof-checking time in cvc5's case.

## 8 Conclusion and Future Work

We presented and evaluated a flexible proof production architecture, showing it is capable of producing proofs with varying levels of granularity in a scalable manner for a state-of-the-art and industrial-strength SMT solver like cvc5.

Since currently, there is no standard proof format for SMT solvers, our architecture is designed to support multiple proof formats via a final post-processing transformation to convert internal proofs accordingly. We are developing backends for the LFSC [52] proof checker and the proof assistants Lean 4 [21], Isabelle/HOL [44], and Coq [10], the latter two via the Alethe proof format [50]. Since using these tools requires mechanizing the respective target proof calculi in their languages, besides external checking, another benefit is to decouple confidence on the soundness of the proof calculi from the internal cvc5 proof calculus.

A considerable challenge for SMT proofs is the plethora of rewrite rules used by the solvers, which are specific for each theory and vary in complexity. In

particular, string rewrites can be very involved [46] and hard to check. We are also developing an SMT-LIB-based DSL for specifying rewrite rules, to be used during proof reconstruction to decompose rewrite steps in terms of them, thus providing more fine-grained proofs for rewriting.

Finally, we plan to incorporate into the proof-production architecture the unsupported theories and features mentioned in Sects. 5.2 and 6, particularly those relevant for solving performance that currently either leave holes in proofs, such as theory pre-processing or non-linear arithmetic reasoning, or that have to be disabled, such as the use of external SAT solvers in the BV theory.

## References

1. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. *J. Log. Algebr. Methods Program.* **119**, 100633 (2021)
2. Abrahám, E., Davenport, J.H., England, M., Kremer, G.: Proving UNSAT in SMT: the case of quantifier free non-linear real arithmetic. *arXiv preprint arXiv:2108.05320* (2021)
3. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to coq through proof witnesses. In: Jouannaud, J.-P., Shao, Z. (eds.) *CPP 2011. LNCS*, vol. 7086, pp. 135–150. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-25379-9\\_12](https://doi.org/10.1007/978-3-642-25379-9_12)
4. Backes, J., et al.: Semantic-based automated reasoning for AWS access policies using SMT. In: Bjørner, N., Gurfinkel, A. (eds.) *Formal Methods in Computer-Aided Design (FMCAD)*, pp. 1–9. IEEE (2018)
5. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*. LNCS, Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-99524-9\\_24](https://doi.org/10.1007/978-3-030-99524-9_24)
6. Barbosa, H., Blanchette, J.C., Fleury, M., Fontaine, P.: Scalable fine-grained proofs for formula processing. *J. Autom. Reason.* **64**(3), 485–510 (2020)
7. Barrett, C., de Moura, L., Fontaine, P.: Proofs in satisfiability modulo theories. *All About Proofs Proofs All (APPA)* **55**(1), 23–44 (2014)
8. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard: version 2.6. Technical report, Department of Computer Science, The University of Iowa (2017). [www.SMT-LIB.org](http://www.SMT-LIB.org)
9. Barrett, C., Tinelli, C.: Satisfiability Modulo Theories. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) *Handbook of Model Checking*, pp. 305–343. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-10575-8\\_11](https://doi.org/10.1007/978-3-319-10575-8_11)
10. Bertot, Y., Castéran, P.: *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series, Springer, Heidelberg (2004). <https://doi.org/10.1007/978-3-662-07964-5>
11. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending sledgehammer with SMT solvers. *J. Autom. Reason.* **51**(1), 109–128 (2013)
12. Böhme, S., Fox, A.C.J., Sewell, T., Weber, T.: Reconstruction of Z3's bit-vector proofs in HOL4 and Isabelle/HOL. In: Jouannaud, J.-P., Shao, Z. (eds.) *CPP 2011. LNCS*, vol. 7086, pp. 183–198. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-25379-9\\_15](https://doi.org/10.1007/978-3-642-25379-9_15)



13. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 179–194. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14052-5\\_14](https://doi.org/10.1007/978-3-642-14052-5_14)
14. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: veriT: an open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 151–156. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02959-2\\_12](https://doi.org/10.1007/978-3-642-02959-2_12)
15. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: an interpolating SMT solver. In: Donaldson, A., Parker, D. (eds.) SPIN 2012. LNCS, vol. 7385, pp. 248–254. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31759-0\\_19](https://doi.org/10.1007/978-3-642-31759-0_19)
16. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Satisfiability modulo transcendental functions via incremental linearization. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 95–113. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63046-5\\_7](https://doi.org/10.1007/978-3-319-63046-5_7)
17. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-36742-7\\_7](https://doi.org/10.1007/978-3-642-36742-7_7)
18. Cohen, C.: Construction of real algebraic numbers in CoQ. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 67–82. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32347-8\\_6](https://doi.org/10.1007/978-3-642-32347-8_6)
19. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 220–236. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63046-5\\_14](https://doi.org/10.1007/978-3-319-63046-5_14)
20. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 118–135. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54577-5\\_7](https://doi.org/10.1007/978-3-662-54577-5_7)
21. de Moura, L., Ullrich, S.: The lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 625–635. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_37](https://doi.org/10.1007/978-3-030-79876-5_37)
22. de Moura, L.M., Bjørner, N.: Proofs and refutations, and Z3. In: Rudnicki, P., Sutcliffe, G., Konev, B., Schmidt, R.A., Schulz, S. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) Workshops. CEUR Workshop Proceedings, vol. 418. CEUR-WS.org (2008)
23. Déharbe, D., Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Exploiting symmetry in SMT problems. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 222–236. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22438-6\\_18](https://doi.org/10.1007/978-3-642-22438-6_18)
24. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006). [https://doi.org/10.1007/11817963\\_11](https://doi.org/10.1007/11817963_11)
25. Eberl, M.: A decision procedure for univariate real polynomials in Isabelle/HOL. In: Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, pp. 75–83. Association for Computing Machinery, New York (2015)
26. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005). [https://doi.org/10.1007/11499107\\_5](https://doi.org/10.1007/11499107_5)
27. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-24605-3\\_37](https://doi.org/10.1007/978-3-540-24605-3_37)



28. Ekici, B., et al.: SMTCoq: a plug-in for integrating SMT solvers into Coq. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 126–133. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_7](https://doi.org/10.1007/978-3-319-63390-9_7)
29. Enderton, H.B.: A Mathematical Introduction to Logic, 2nd edn. Academic Press, Cambridge (2001)
30. Farkas, G.: A Fourier-féle mechanikai elv alkalmazásai. *Mathematikai és Természettudományi Értesítő* **12**, 457–472 (1894). Reference from Schrijver’s Combinatorial Optimization textbook (Hungarian)
31. Fontaine, P., Marion, J.-Y., Merz, S., Nieto, L.P., Tiu, A.: Expressiveness + automation + soundness: towards combining SMT solvers and interactive proof assistants. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 167–181. Springer, Heidelberg (2006). [https://doi.org/10.1007/11691372\\_11](https://doi.org/10.1007/11691372_11)
32. Hadarean, L., Barrett, C., Reynolds, A., Tinelli, C., Deters, M.: Fine grained SMT proofs for the theory of fixed-width bit-vectors. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 340–355. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48899-7\\_24](https://doi.org/10.1007/978-3-662-48899-7_24)
33. Heule, M.J.H.: The DRAT format and drat-trim checker. CoRR, abs/1610.06229 (2016)
34. Joosten, S.J.C., Thiemann, R., Yamada, A.: A verified implementation of algebraic numbers in Isabelle/HOL. *J. Autom. Reason.* **64**, 363–389 (2020)
35. Kan, S., Lin, A.W., Rümmer, P., Schrader, M.: Certistr: a certified string solver. In: Popescu, A., Zdancewic, S. (eds.) Certified Programs and Proofs (CPP), pp. 210–224. ACM (2022)
36. Katz, G., Barrett, C., Tinelli, C., Reynolds, A., Hadarean, L.: Lazy proofs for DPLL(T)-based SMT solvers. In: Piskac, R., Talupur, M. (eds.) Formal Methods in Computer-Aided Design (FMCAD), pp. 93–100. IEEE (2016)
37. Kiesl, B., Rebola-Pardo, A., Heule, M.J.H.: Extended resolution simulates DRAT. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 516–531. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-94205-6\\_34](https://doi.org/10.1007/978-3-319-94205-6_34)
38. King, T.: Effective algorithms for the satisfiability of quantifier-free formulas over linear real and integer arithmetic (2014)
39. Lammich, P.: Efficient verified (UN)SAT certificate checking. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 237–254. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63046-5\\_15](https://doi.org/10.1007/978-3-319-63046-5_15)
40. Liang, T., Reynolds, A., Tinelli, C., Barrett, C., Deters, M.: A DPLL(T) theory solver for a theory of strings and regular expressions. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 646–662. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08867-9\\_43](https://doi.org/10.1007/978-3-319-08867-9_43)
41. Mahboubi, A.: Implementing the cylindrical algebraic decomposition within the coq system. *Math. Struct. Comput. Sci.* **17**(1), 99–127 (2007)
42. Nieuwenhuis, R., Oliveras, A.: Proof-producing congruence closure. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 453–468. Springer, Heidelberg (2005). [https://doi.org/10.1007/978-3-540-32033-3\\_33](https://doi.org/10.1007/978-3-540-32033-3_33)
43. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *J. ACM* **53**(6), 937–977 (2006)
44. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45949-9>

45. Ozdemir, A., Niemetz, A., Preiner, M., Zohar, Y., Barrett, C.: DRAT-based bit-vector proofs in CVC4. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 298–305. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-24258-9\\_21](https://doi.org/10.1007/978-3-030-24258-9_21)
46. Reynolds, A., Nötzli, A., Barrett, C., Tinelli, C.: High-level abstractions for simplifying extended string constraints in SMT. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11562, pp. 23–42. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-25543-5\\_2](https://doi.org/10.1007/978-3-030-25543-5_2)
47. Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.: Reductions for strings and regular expressions revisited. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 225–235. IEEE (2020)
48. Reynolds, A., Woo, M., Barrett, C., Brumley, D., Liang, T., Tinelli, C.: Scaling up DPLL(T) string solvers using context-dependent simplification. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 453–474. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_24](https://doi.org/10.1007/978-3-319-63390-9_24)
49. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Hoboken (1998)
50. Schurr, H.-J., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: towards a generic SMT proof format (extended abstract). CoRR, abs/2107.02354 (2021)
51. Schurr, H.-J., Fleury, M., Desharnais, M.: Reliable reconstruction of fine-grained proofs in a proof assistant. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 450–467. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_26](https://doi.org/10.1007/978-3-030-79876-5_26)
52. Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods Syst. Des. **42**(1), 91–118 (2013)
53. Thiemann, R., Yamada, A.: Algebraic numbers in Isabelle/HOL. In: Blanchette, J.C., Merz, S. (eds.) ITP 2016. LNCS, vol. 9807, pp. 391–408. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-43144-4\\_24](https://doi.org/10.1007/978-3-319-43144-4_24)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# CTL\* Model Checking for Data-Aware Dynamic Systems with Arithmetic

Paolo Felli, Marco Montali, and Sarah Winkler<sup>(✉)</sup>

Free University of Bolzano-Bozen, Bolzano, Italy  
{pfelli,montali,winkler}@inf.unibz.it

**Abstract.** The analysis of complex dynamic systems is a core research topic in formal methods and AI, and combined modelling of systems with data has gained increasing importance in applications such as business process management. In addition, process mining techniques are nowadays used to automatically mine process models from event data, often without correctness guarantees. Thus verification techniques for linear and branching time properties are needed to ensure desired behavior.

Here we consider data-aware dynamic systems with arithmetic (DDSAs), which constitute a concise but expressive formalism of transition systems with linear arithmetic guards. We present a CTL\* model checking procedure for DDSAs that addresses a generalization of the classical verification problem, namely to compute conditions on the initial state, called *witness maps*, under which the desired property holds. Linear-time verification was shown to be decidable for specific classes of DDSAs where the constraint language or the control flow are suitably confined. We investigate several of these restrictions for the case of CTL\*, with both positive and negative results: witness maps can always be found for monotonicity and integer periodicity constraint systems, but verification of bounded lookback systems is undecidable. To demonstrate the feasibility of our approach, we implemented it in an SMT-based prototype, showing that many practical business process models can be effectively analyzed.

**Keywords:** Verification · CTL\* · Counter systems · Constraints · SMT

## 1 Introduction

The study of complex dynamic systems is a core research topic in AI, with a long tradition in formal methods. It finds application in a variety of domains, such as notably business process management (BPM), where studying the interplay between control-flow and data has gained momentum [9, 10, 24, 46]. Processes are increasingly mined by automatic techniques [1, 3] that lack any correctness guarantees, making verification even more important to ensure the desired behavior.

---

This work is partially supported by the UNIBZ projects DaCoMan, QUEST, SMART-APP, VERBA, and WineId.

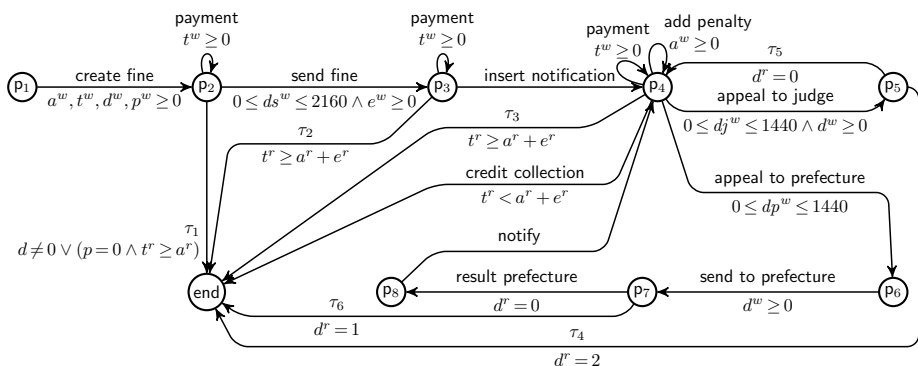
© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 36–56, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_4](https://doi.org/10.1007/978-3-031-10769-6_4)

In this work, we focus on the concise but expressive framework of data-aware dynamic systems with arithmetic (DDSAs) [28,38], also known as counter systems [13,20,34]. Several classes of DDSAs have been isolated where specific verification tasks are decidable, notably reachability [6,13,29,34] and linear-time model checking [14,20,22,28,38]. Fewer results are known about the case of branching time, except for flat counter systems [21], gap-order systems where constraints are restricted to the form  $x - y \geq 2$  [8,42], and systems with a *nice symbolic valuation abstraction* [31]. However, many processes in BPM and beyond fall into neither of these classes, as illustrated by the next example.

*Example 1.* The following DDSA  $\mathcal{B}$  models a management process for road fines by the Italian police [41]. It maintains seven so-called *case data* variables (i.e. variables local to each process instance, called “case” in the BPM literature):  $a$  (amount),  $t$  (total amount),  $d$  (dismissal code),  $p$  (points deducted),  $e$  (expenses), and time durations  $ds$ ,  $dp$ ,  $dj$ . The process starts by creating a case, upon which the offender is notified within 90 days, i.e., 2160h (**send fine**). If the offender pays a sufficient amount  $t$ , the process terminates via silent actions  $\tau_1$ ,  $\tau_2$ , or  $\tau_3$ . For the less happy paths, the **credit collection** action is triggered if the payment was insufficient; while **appeal to judge** and **appeal to prefecture** reflect filed protests by the offender, which again need to respect certain time constraints.



This model was generated from real-life logs by automatic process mining techniques paired with domain knowledge [41], but without any correctness guarantee. For instance, *data-aware soundness* [4, 25] requires that the process can always reach a final state from any reachable configuration, expressed by the branching-time property  $\text{AGEF end}$ . This property is false here, as  $\mathcal{B}$  can get stuck in state  $\mathbf{p}_7$  if  $d > 1$ . In addition, process-specific linear-time properties are needed, e.g., that a **send fine** event is always followed by a sufficient payment (i.e.,  $\langle \text{send fine} \rangle \top \rightarrow \mathbf{F} \langle \text{payment} \rangle (t \geq a)$ , where  $\langle \alpha \rangle$  is the next operator via action  $\alpha$ ).

This example highlights how both linear-time and branching-time verification are needed. In this paper, we present a CTL\* model checking algorithm for DDSAs, adopting a finite-trace semantics ( $\text{CTL}_f^*$ ) [44] to reflect the nature of processes as in Example 1. More precisely, our approach can synthesize conditions on the initial variable assignment such that a given property  $\chi$  holds, called *witness maps*. If such a witness map can be found, it is in particular decidable what is more commonly called the *verification problem*, namely whether  $\chi$  is satisfied in a designated initial configuration. We derive an abstract criterion on the computability of witness maps, which is satisfied by two practical DDSA classes that restrict the constraint language to (a) monotonicity constraints [20, 25], i.e., variable-to-variable or variable-to-constant comparisons over  $\mathbb{Q}$  or  $\mathbb{R}$ , and (b) integer periodicity constraints [18, 22], i.e., variable-to-constant and restricted variable-to-variable comparisons with modulo operators. On the other hand, we show that the verification problem is undecidable for *bounded lookback* systems [28], a control flow restriction that generalizes *feedback freedom* [14].

In summary, we make the following contributions:

1. We present a model checking algorithm to generate a witness map for a given DDSA and  $\text{CTL}_f^*$  property;
2. We prove an abstract termination criterion for this algorithm (Corollary 1);
3. This result is used to show that witness maps can be effectively computed for monotonicity constraint and integer periodicity constraint systems;
4.  $\text{CTL}_f^*$  verification is shown undecidable for bounded-lookback systems;
5. We implemented our approach in the prototype `ada` using SMT solvers as backends and tested it on a range of business processes from the literature.

The paper is structured as follows: The rest of this section recapitulates related work. Section 2 compiles preliminaries about DDSAs and  $\text{CTL}_f^*$ . Section 3 is dedicated to LTL with configuration maps, which is used by our model checking procedure in Sect. 4. Based on an abstract termination criterion, (un)decidability results for concrete DDSA classes are given in Sect. 5. We describe our implementation in Sect. 6. Complete proofs and further examples can be found in [27].

*Related work.* Verification of transition systems with arithmetic constraints, also called counter systems, has been studied in many areas including formal methods, database theory, and BPM. Reachability was proven decidable for a variety of classes, e.g., reversal-bounded counter machines [34], finite linear [29], flat [13], and gap-order constraint (GC) systems [6]. Considerable work has also been dedicated to linear-time verification: LTL model checking is decidable for monotonicity constraint (MC) systems [20]. LTL verification is also decidable for integer periodicity constraint (IPC) systems, even with past-time operators [18, 22]; and feedback-free systems, for an enriched constraint language referring to a read-only database [14]. DDSAs with MCs are also considered in [25] from the perspective of LTL with a finite-run semantics ( $\text{LTL}_f$ ), giving a procedure to compute finite, faithful abstractions.  $\text{LTL}_f$  is moreover decidable for systems with the abstract *finite summary* property [28], which includes MC, GC, and systems with bounded lookback, where the latter generalizes feedback freedom.

Branching-time verification was less studied: Decidability of CTL\* was proven for flat counter systems with Presburger-definable loop iteration [21], even in NP [19]. Moreover, it was shown that CTL\* verification is decidable for pushdown systems, which can model counter systems with a single integer variable [30]. For integer relational automata (IRA), i.e., systems with constraints  $x \geq y$  or  $x > y$  and domain  $\mathbb{Z}$ , CTL model checking is undecidable while the existential and universal fragments of CTL\* remain decidable [12]. For GC systems, which extend IRAs to constraints of the form  $x - y \geq k$ , the existential fragment of CTL\* is decidable while the universal one is not [8]. A similar dichotomy holds for the EF and EG fragments of CTL [42]. A subclass of IRAs was considered in [7, 11], allowing only periodicity and monotonicity constraints. While satisfiability of CTL\* was proven decidable, model checking is not (as already shown in [12]), though it is decidable for CEF<sup>+</sup> properties, an extension of the EF fragment [7]. In contrast, rather than restricting temporal operators, we show decidability of model checking under an abstract property of the DDSA and the verified property, which can be guaranteed by suitably constraining the constraint class or the control flow. More closely related is work by Gascon [31], who shows decidability of CTL\* model checking for DDSAs that admit a *nice symbolic valuation abstraction*, an abstract property which includes MC and IPC systems. The relationship between our decidability criterion and the property defined by Gascon will need further investigation. Another difference is that we here adopt a finite-path semantics for CTL\* as e.g. considered in [47], since for the analysis of real-world processes such as business processes it is sufficient to consider finite traces. On a high level, our method follows a common approach to CTL\*: the verification property is processed bottom-up, computing solutions for each subproperty. These are then used to solve an equivalent linear-time problem [2, p. 429]. For the latter, we partially rely on earlier work [28].

## 2 Background

We start by defining the set of constraints over expressions of sort *int*, *rat*, or *real*, with associated domains  $\text{dom}(\text{int}) = \mathbb{Z}$ ,  $\text{dom}(\text{rat}) = \mathbb{Q}$ , and  $\text{dom}(\text{real}) = \mathbb{R}$ .

**Definition 1.** *For a given set of sorted variables  $V$ , expressions  $e_s$  of sort  $s$  and atoms  $a$  are defined as follows:*

$$e_s := v_s \mid k_s \mid e_s + e_s \mid e_s - e_s \quad a := e_s = e_s \mid e_s < e_s \mid e_s \leq e_s \mid e_{\text{int}} \equiv_n e_{\text{int}}$$

where  $k_s \in \text{dom}(s)$ ,  $v_s \in V$  has sort  $s$ , and  $\equiv_n$  denotes equality modulo some  $n \in \mathbb{N}$ . A constraint is then a quantifier-free boolean expression over atoms  $a$ .

The set of all constraints built from atoms over variables  $V$  is denoted by  $\mathcal{C}(V)$ . For instance,  $x \neq 1$ ,  $x < y - z$ , and  $x - y = 2 \wedge y \neq 1$  are valid constraints independent of the sort of  $\{x, y, z\}$ , while  $u \equiv_3 v + 1$  is a constraint for integer variables  $u$  and  $v$ . We write  $\mathcal{V}\text{ar}(\varphi)$  for the set of variables in a formula  $\varphi$ . For

an assignment  $\alpha$  with domain  $V$  that maps variables to values in their domain, and a formula  $\varphi$  we write  $\alpha \models \varphi$  if  $\alpha$  satisfies  $\varphi$ .

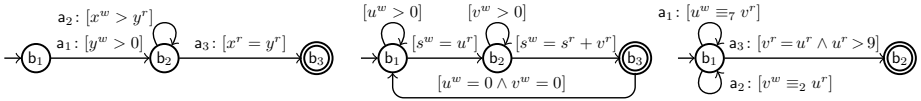
We are thus in the realm of SMT with linear arithmetic, which is decidable and admits *quantifier elimination* [45]: if  $\varphi$  is a formula in  $\mathcal{C}(X \cup \{y\})$ , thus having free variables  $X \cup \{y\}$ , there is a quantifier-free  $\varphi'$  with free variables  $X$  that is equivalent to  $\exists y.\varphi$ , i.e.,  $\varphi' \equiv \exists y.\varphi$ , where  $\equiv$  denotes logical equivalence.

## 2.1 Data-Aware Dynamic Systems with Arithmetic

From now on,  $V$  is a fixed, finite set of variables. We consider two disjoint, marked copies of  $V$ , denoted  $V^r = \{v^r \mid v \in V\}$  and  $V^w = \{v^w \mid v \in V\}$ , called the *read* and *write* variables. They will refer to variable values before and after a transition, respectively. We also write  $\bar{V}$  for a vector that orders  $V$  in an arbitrary but fixed way, and  $\bar{V}^r$  and  $\bar{V}^w$  for vectors ordering  $V^r$  and  $V^w$  in the same way.

**Definition 2.** A DDSA  $\mathcal{B} = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, \text{guard} \rangle$  is a labeled transition system where (i)  $B$  is a finite set of control states, with  $b_I \in B$  the initial one; (ii)  $\mathcal{A}$  is a set of actions; (iii)  $T \subseteq B \times \mathcal{A} \times B$  is a transition relation; (iv)  $B_F \subseteq B$  are final states; (v)  $V$  is the set of process variables; (vi)  $\alpha_I$  the initial variable assignment; (vii)  $\text{guard}: \mathcal{A} \mapsto \mathcal{C}(V^r \cup V^w)$  specifies executability constraints for actions over variables  $V^r \cup V^w$ .

*Example 2.* We consider the following DDSAs  $\mathcal{B}$ ,  $\mathcal{B}_{bl}$ , and  $\mathcal{B}_{ipc}$ , where  $x, y$  have domain  $\mathbb{Q}$  and  $u, v, s$  have domain  $\mathbb{Z}$ . Initial and final states have incoming arrows and double borders, respectively;  $\alpha_I$  is not fixed for now.



Also the system in Example 1 represents a DDSA. If state  $b$  admits a transition to  $b'$  via action  $a$ , namely  $(b, a, b') \in \Delta$ , this is denoted by  $b \xrightarrow{a} b'$ . A *configuration* of  $\mathcal{B}$  is a pair  $(b, \alpha)$  where  $b \in B$  and  $\alpha$  is an assignment with domain  $V$ . A *guard assignment* is an assignment  $\beta$  with domain  $V^r \cup V^w$ . For an action  $a$ , let  $\text{write}(a) = \text{Var}(\text{guard}(a)) \cap V^w$ . As defined next, an action  $a$  transforms a configuration  $(b, \alpha)$  into a new configuration  $(b', \alpha')$  by updating the assignment  $\alpha$  according to the action guard, which can at the same time evaluate conditions on the current values of variables and write new values:

**Definition 3.** A DDSA  $\mathcal{B} = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, \text{guard} \rangle$  admits a step from configuration  $(b, \alpha)$  to  $(b', \alpha')$  via action  $a$ , denoted  $(b, \alpha) \xrightarrow{a} (b', \alpha')$ , if  $b \xrightarrow{a} b'$ ,  $\alpha'(v) = \alpha(v)$  for all  $v \in V \setminus \text{write}(a)$ , and the guard assignment  $\beta$  given by  $\beta(v^r) = \alpha(v)$  and  $\beta(v^w) = \alpha'(v)$  for all  $v \in V$ , satisfies  $\beta \models \text{guard}(a)$ .

For instance, for  $\mathcal{B}$  in Example 2 and initial assignment  $\alpha_I(x) = \alpha_I(y) = 0$ , the initial configuration admits a step  $(b_1, \begin{bmatrix} x=0 \\ y=0 \end{bmatrix}) \xrightarrow{a_1} (b_2, \begin{bmatrix} x=0 \\ y=3 \end{bmatrix})$  with  $\beta(x^r) = \beta(x^w) = \beta(y^r) = 0$  and  $\beta(y^w) = 3$ .



A *run*  $\rho$  of a DDSA  $\mathcal{B}$  of length  $n$  from configuration  $(b, \alpha)$  is a sequence of steps  $\rho: (b, \alpha) = (b_0, \alpha_0) \xrightarrow{a_1} (b_1, \alpha_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (b_n, \alpha_n)$ . We also associate with  $\rho$  the *symbolic run*  $\sigma: b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} b_n$  where state and action sequences are recorded without assignments, and say that  $\sigma$  is the *abstraction* of  $\rho$  (or,  $\sigma$  *abstracts*  $\rho$ ). For some  $m < n$ ,  $\sigma|_m$  denotes the prefix of  $\sigma$  that has  $m$  steps.

## 2.2 History Constraints

In this section, we fix a DDSA  $\mathcal{B} = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, \text{guard} \rangle$ . We aim to build an abstraction of  $\mathcal{B}$  that covers the (potentially infinite) set of configurations by finitely many nodes of the form  $(b, \varphi)$ , where  $b \in B$  is a control state and  $\varphi$  a formula that expresses conditions on the variables  $V$ . A state  $(b, \varphi)$  thus represents all configurations  $(b, \alpha)$  s.t.  $\alpha \models \varphi$ . To express how such a formula  $\varphi$  is modified by executing an action, let the *transition formula* of action  $a$  be  $\Delta_a(\bar{V}^r, \bar{V}^w) = \text{guard}(a) \wedge \bigwedge_{v \in V \setminus \text{write}(a)} v^w = v^r$ . This states conditions on variables before and after executing  $a$ :  $\text{guard}(a)$  must hold and the values of all variables that are not written are propagated by inertia. We write  $\Delta_a(\bar{X}, \bar{Y})$  for the formula obtained from  $\Delta_a$  by replacing  $\bar{V}^r$  by  $\bar{X}$  and  $\bar{V}^w$  by  $\bar{Y}$ . Let a variable vector  $\bar{U}$  be a *fresh copy* of  $\bar{V}$  if it has the same length as  $|\bar{V}|$  and  $U \cap V = \emptyset$ . To mimic steps on the abstract level, we define the following *update* function:

**Definition 4.** For a formula  $\varphi$  with free variables  $V$  and action  $a$ ,  $\text{update}(\varphi, a) = \exists \bar{U}. \varphi(\bar{U}) \wedge \Delta_a(\bar{U}, \bar{V})$ , where  $\bar{U}$  is a fresh copy of  $\bar{V}$ .

Our approach will generate formulas of a special shape called *history constraints* [28], obtained by iterated *update* operations in combination with a sequence of *verification constraints*  $\bar{\vartheta}$ . Intuitively, the latter depends on the verification property. For now it suffices to consider  $\bar{\vartheta}$  an arbitrary sequence of constraints with free variables  $V$ . Its prefix of length  $k$  is denoted by  $\bar{\vartheta}|_k$ . We need a fixed set of placeholder variables  $V_0$  disjoint from  $V$ , and assume an injective variable renaming  $\nu: V \mapsto V_0$ . Let  $\varphi_\nu$  be the formula  $\varphi_\nu = \bigwedge_{v \in V} v = \nu(v)$ .

**Definition 5.** For a symbolic run  $\sigma: b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} b_n$ , and verification constraint sequence  $\bar{\vartheta} = \langle \vartheta_0, \dots, \vartheta_n \rangle$ , the history constraint  $h(\sigma, \bar{\vartheta})$  is given by  $h(\sigma, \bar{\vartheta}) = \varphi_\nu \wedge \vartheta_0$  if  $n = 0$ , and  $h(\sigma, \bar{\vartheta}) = \text{update}(h(\sigma|_{n-1}, \bar{\vartheta}|_{n-1}), a_n) \wedge \vartheta_n$  if  $n > 0$ .

Thus, history constraints are formulas with free variables  $V \cup V_0$ . Satisfying assignments for history constraints are closely related to assignments in runs:<sup>1</sup>

**Lemma 1.** For a symbolic run  $\sigma: b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} b_n$  and  $\bar{\vartheta} = \langle \vartheta_0, \dots, \vartheta_n \rangle$ ,  $h(\sigma, \bar{\vartheta})$  is satisfied by assignment  $\alpha$  with domain  $V \cup V_0$  iff  $\sigma$  abstracts a run  $\rho: (b_0, \alpha_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (b_n, \alpha_n)$  such that (i)  $\alpha_0(v) = \alpha(\nu(v))$ , and (ii)  $\alpha_n(v) = \alpha(v)$  for all  $v \in V$ , and (iii)  $\alpha_i \models \vartheta_i$  for all  $i$ ,  $0 \leq i \leq n$ .

<sup>1</sup> Lemma 1 is a slight variation of [28, Lemma 3.5]: Definition 5 differs from history constraints in [28] in that the initial assignment is not fixed. A proof can be found in [27].



### 2.3 CTL<sub>f</sub>\*

For a DDSA  $\mathcal{B}$  as above, we consider the following verification properties:

**Definition 6.** *CTL<sub>f</sub>\* state formulas  $\chi$  and path formulas  $\psi$  are defined by the following grammar, for constraints  $c \in \mathcal{C}(V)$  and control states  $b \in B$ :*

$$\chi := \top \mid c \mid b \mid \chi \wedge \chi \mid \neg \chi \mid \mathbf{E} \psi \quad \psi := \chi \mid \psi \wedge \psi \mid \neg \psi \mid \mathbf{X} \psi \mid \mathbf{G} \psi \mid \psi \mathbf{U} \psi$$

We use the usual abbreviations  $\mathbf{F} \psi = \top \mathbf{U} \psi$ ,  $\chi_1 \vee \chi_2 = \neg(\neg \chi_1 \wedge \neg \chi_2)$ , and  $\mathbf{A} \psi = \neg \mathbf{E} \neg \psi$ . To simplify the presentation, we do not explicitly treat next state operators  $\langle a \rangle$  via a specific action  $a$ , as used in Example 1, though this would be possible (cf. [28]). However, such an operator can be encoded by adding a fresh data variable  $x$  to  $V$ , the conjunct  $x^w = 1$  to  $\text{guard}(a)$ , and  $x^w = 0$  to all other guards, and replacing  $\langle a \rangle \psi$  in the verification property by  $\mathbf{X}(\psi \wedge x = 1)$ .

The maximal number of nested path quantifiers in a formula  $\psi$  is called the *quantifier depth* of  $\psi$ , denoted by  $qd(\psi)$ . We adopt a finite path semantics for CTL\* [44]: For a control state  $b \in B$  and a state assignment  $\alpha$ , let  $FRuns(b, \alpha)$  be the set of *final runs*  $\rho: (b, \alpha) = (b_0, \alpha_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (b_n, \alpha_n)$  such that  $b_n \in F$  is a final state. The  $i$ -th configuration  $(b_i, \alpha_i)$  in  $\rho$  is denoted by  $\rho_i$ .

**Definition 7.** *The semantics of CTL<sub>f</sub>\* is inductively defined as follows. For a DDSA  $\mathcal{B}$  with configuration  $(b, \alpha)$ , state formulas  $\chi, \chi'$ , and path formulas  $\psi, \psi'$ :*

$$\begin{aligned} (b, \alpha) &\models \top \\ (b, \alpha) &\models c \quad \text{iff } \alpha \models c \\ (b, \alpha) &\models b' \quad \text{iff } b = b' \\ (b, \alpha) &\models \chi \wedge \chi' \quad \text{iff } (b, \alpha) \models \chi \text{ and } (b, \alpha) \models \chi' \\ (b, \alpha) &\models \neg \chi \quad \text{iff } (b, \alpha) \not\models \chi \\ (b, \alpha) &\models \mathbf{E} \psi \quad \text{iff } \exists \rho \in FRuns(b, \alpha) \text{ such that } \rho \models \psi \end{aligned}$$

where  $\rho \models \psi$  iff  $\rho, 0 \models \psi$  holds, and for a run  $\rho$  of length  $n$  and all  $i$ ,  $0 \leq i \leq n$ :

$$\begin{aligned} \rho, i &\models \chi \quad \text{iff } \rho_i \models \chi \\ \rho, i &\models \neg \psi \quad \text{iff } \rho, i \not\models \psi \\ \rho, i &\models \psi \wedge \psi' \quad \text{iff } \rho, i \models \psi \text{ and } \rho, i \models \psi' \\ \rho, i &\models \mathbf{X} \psi \quad \text{iff } i < n \text{ and } \rho, i + 1 \models \psi \\ \rho, i &\models \mathbf{G} \psi \quad \text{iff for all } j, i \leq j \leq n, \text{ it holds that } \rho, j \models \psi \\ \rho, i &\models \psi \mathbf{U} \psi' \quad \text{iff } \exists k \text{ with } i + k \leq n \text{ such that } \rho, i + k \models \psi' \\ &\quad \text{and for all } j, 0 \leq j < k, \text{ it holds that } \rho, i + j \models \psi. \end{aligned}$$

Instead of simply checking whether the initial configuration of a DDSA  $\mathcal{B}$  satisfies a CTL<sub>f</sub>\* property  $\chi$ , we try to determine, for every state  $b \in B$ , which constraints on variables need to hold in order to satisfy  $\chi$ . As the number of configurations  $(b, \alpha)$  of a DDSA  $\mathcal{B}$  is usually infinite, configuration sets cannot be enumerated explicitly. Instead, we represent a set of configurations as a *configuration map*  $K: B \mapsto \mathcal{C}(V)$  that associates with every control state  $b \in B$  a formula  $K(b) \in \mathcal{C}(V)$ , representing all configurations  $(b, \alpha)$  such that  $\alpha \models K(b)$ .

We now define when a configuration captures the maximal set of configurations in which a formula  $\chi$  holds. We call these witness maps.

**Definition 8.** For a DDSA  $\mathcal{B}$  and state formula  $\chi$ , a configuration map  $K$  is a witness map if it holds that  $(b, \alpha) \models \chi$  iff  $\alpha \models K(b)$ , for all  $b \in B$  and all  $\alpha$ .

For instance, for  $\mathcal{B}$  from Example 2 and  $\chi_1 = \text{AG}(x \geq 2)$ , a witness map is given by  $K = \{b_1 \mapsto \perp, b_2 \mapsto x \geq 2 \wedge y \geq 2, b_3 \mapsto x \geq 2\}$ . For  $\chi_2 = \text{EX}(\text{AG}(x \geq 2))$ , a solution is  $K' = \{b_1 \mapsto x \geq 2, b_2 \mapsto y \geq 2, b_3 \mapsto \perp\}$ . As  $b_1$  is the initial state,  $\mathcal{B}$  satisfies  $\chi_2$  with every initial assignment that sets  $\alpha_I(x) \geq 2$ .

In this paper we address the problem of finding a witness map for  $\mathcal{B}$  and  $\chi$ . Note that a witness map in particular allows to decide what is commonly called the *verification problem*, namely to check whether  $(b_I, \alpha_I) \models \chi$  holds, by testing  $\alpha_I \models K(b_I)$ . It remains to investigate whether there exist a DDSA  $\mathcal{B}$  and  $\chi$  for which no witness map exists, as the configuration set satisfying  $\chi$  is not finitely representable. Even if it exists, finding it is in general undecidable. However, in this paper we identify DDSA classes where a witness map can always be found.

### 3 LTL with Configuration Maps

Following a common approach to CTL\* verification, our technique processes the property  $\chi$  bottom-up, computing solutions for each subformula  $\text{E}\psi$ , before solving a linear-time model checking problem  $\chi'$  in which the solutions to subformulas appear as atoms. Given our representation of sets of configurations, we use LTL formulas where atoms are configuration maps, and denote this specification language by  $\text{LTL}_f^{\mathcal{B}}$ . For a given DDSA  $\mathcal{B}$ , it is formally defined as follows:

$$\psi := K \mid \psi \wedge \psi' \mid \neg\psi \mid \text{X}\psi \mid \text{G}\psi \mid \psi \text{ U } \psi'$$

where  $K \in \mathcal{K}_{\mathcal{B}}$ , for  $\mathcal{K}_{\mathcal{B}}$  is the set of configuration maps for  $\mathcal{B}$ .

**Definition 9.** A run  $\rho$  of length  $n$  satisfies an  $\text{LTL}_f^{\mathcal{B}}$  formula  $\psi$ , denoted  $\rho \models_{\mathcal{K}} \psi$ , iff  $\rho, 0 \models_{\mathcal{K}} \psi$  holds, where for all  $i$ ,  $0 \leq i \leq n$ :

$$\begin{aligned} \rho, i \models_{\mathcal{K}} K & \quad \text{iff } \rho_i = (b, \alpha) \text{ and } \alpha \models K(b); \\ \rho, i \models_{\mathcal{K}} \psi \wedge \psi' & \quad \text{iff } \rho, i \models_{\mathcal{K}} \psi \text{ and } \rho, i \models_{\mathcal{K}} \psi'; \\ \rho, i \models_{\mathcal{K}} \neg\psi & \quad \text{iff } \rho, i \not\models_{\mathcal{K}} \psi; \\ \rho, i \models_{\mathcal{K}} \text{X}\psi & \quad \text{iff } i < n \text{ and } \rho, i+1 \models_{\mathcal{K}} \psi; \\ \rho, i \models_{\mathcal{K}} \text{G}\psi & \quad \text{iff } \rho, i \models_{\mathcal{K}} \psi \text{ and } (i = n \text{ or } \rho, i+1 \models_{\mathcal{K}} \text{G}\psi); \\ \rho, i \models_{\mathcal{K}} \psi \text{ U } \psi' & \quad \text{iff } \rho, i \models_{\mathcal{K}} \psi' \text{ or } (i < n \text{ and } \rho, i \models_{\mathcal{K}} \psi \text{ and } \rho, i+1 \models_{\mathcal{K}} \psi \text{ U } \psi'). \end{aligned}$$

Our approach to  $\text{LTL}_f^{\mathcal{B}}$  verification proceeds along the lines of the  $\text{LTL}_f$  procedure from [28], with the difference that simple constraint atoms are replaced by configuration maps. In order to express the requirements on a run of a DDSA  $\mathcal{B}$  to satisfy an  $\text{LTL}_f^{\mathcal{B}}$  formula  $\chi$ , we use a nondeterministic automaton (NFA)  $\mathcal{N}_{\psi} = (Q, \Sigma, \varrho, q_0, Q_F)$ , where the states  $Q$  are a set of subformulas of  $\psi$ ,  $\Sigma = 2^{\mathcal{K}_{\mathcal{B}}}$  is the alphabet,  $\varrho$  is the transition relation,  $q_0 \in Q$  is the initial state, and  $Q_F \subseteq Q$  is the set of final states. The construction of  $\mathcal{N}_{\psi}$  is standard [15, 28], treating configuration maps for the time being as propositions; but for completeness it is described in [27, Appendix C]. For instance, for a configuration map  $K$ ,  $\psi = \text{F}K$

corresponds to the NFA  $\rightarrow(\psi) \xrightarrow{K} (\top)$  and  $\psi' = \mathsf{X} K$  to  $\rightarrow(\psi') \xrightarrow{K} (\top)$ . (For simplicity, edges labels  $\{K\}$  are shown as  $K$ , and edge labels  $\emptyset$  are omitted.)

For  $w_i \in \Sigma$ , i.e.,  $w_i$  is a set of configuration maps,  $w_i(b)$  denotes the formula  $\bigwedge_{K \in w_i} K(b)$ . Moreover, for  $w = w_0, \dots, w_n \in \Sigma^*$  and a symbolic run  $\sigma: b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} b_n$ , let  $w \otimes \sigma$  denote the sequence of formulas  $\langle w_0(b_0), \dots, w_n(b_n) \rangle$ , i.e., the component-wise application of  $w$  to the control states of  $\sigma$ . A word  $w_0, \dots, w_n \in \Sigma^*$  is *consistent* with a run  $(b_0, \alpha_0) \xrightarrow{a_1} (b_1, \alpha_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (b_n, \alpha_n)$  if  $\alpha_i \models w_i(b_i)$  for all  $i$ ,  $0 \leq i \leq n$ . The key correctness property of  $\mathcal{N}_\psi$  is the following (cf. [28, Lemma 4.4], and see [27] for the proof adapted to  $\text{LTL}_f^B$ ):

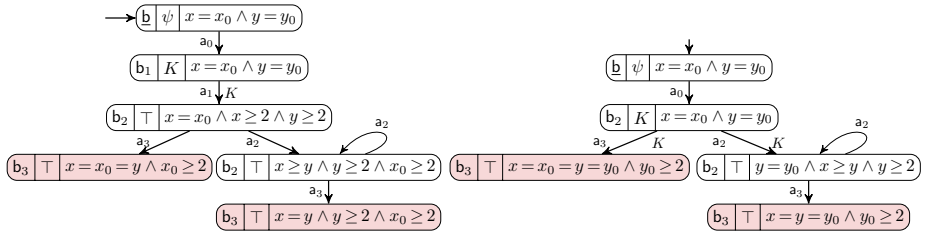
**Lemma 2.**  $\mathcal{N}_\psi$  accepts a word that is consistent with a run  $\rho$  iff  $\rho \models_K \psi$ .

*Product Construction.* As a next step in our verification procedure, given a control state  $b$  of  $\mathcal{B}$ , we aim to find (a symbolic representation of) all configurations  $(b, \alpha)$  that satisfy an  $\text{LTL}_f^B$  formula  $\psi$ . To that end, we combine  $\mathcal{N}_\psi$  with  $\mathcal{B}$  to a cross-product automaton  $\mathcal{N}_{\mathcal{B},b}^\psi$ . For technical reasons, when performing the product construction, the steps in  $\mathcal{B}$  need to be shifted by one with respect to the steps in  $\mathcal{N}_\psi$ . Hence, given  $b \in B$ , let  $\mathcal{B}_b$  be the DDSA obtained from  $\mathcal{B}$  by adding a dummy initial state  $\underline{b}$ , so that  $\mathcal{B}_b$  has state set  $B' = B \cup \{\underline{b}\}$  and transition relation  $T' = T \cup \{(\underline{b}, a_0, b)\}$  for a fresh action  $a_0$  with  $\text{guard}(a_0) = \top$ .

**Definition 10.** The product automaton  $\mathcal{N}_{\mathcal{B},b}^\psi$  is defined for an  $\text{LTL}_f^B$  formula  $\psi$ , a DDSA  $\mathcal{B}$ , and a control state  $b \in B$ . Let  $\mathcal{B}_b = \langle B', \underline{b}, \mathcal{A}, T', B_F, V, \alpha_I, \text{guard} \rangle$  and  $\mathcal{N}_\psi$  as above. Then  $\mathcal{N}_{\mathcal{B},b}^\psi = (P, R, p_0, P_F)$  is as follows:

- $P \subseteq B' \times Q \times \mathcal{C}(V \cup V_0)$ , i.e., states in  $P$  are triples  $(b, q, \varphi)$  such that
- the initial state is  $p_0 = (\underline{b}, q_0, \varphi_\nu)$ ;
- if  $b \xrightarrow{a} b'$  in  $T'$ ,  $q \xrightarrow{w} q'$  in  $\mathcal{N}_\psi$ , and  $\text{update}(\varphi, a) \wedge w(b')$  is satisfiable, there is a transition  $(b, q, \varphi) \xrightarrow{a, w} (b', q', \varphi')$  in  $R$  such that  $\varphi' \equiv \text{update}(\varphi, a) \wedge w(b')$ ;
- $(b', q', \varphi')$  is in the set of final states  $P_F \subseteq P$  iff  $b' \in B_F$ , and  $q' \in Q_F$ .

*Example 3.* Consider the DDSA  $\mathcal{B}$  from Example 2, and let  $K = \{b_1 \mapsto \perp, b_2 \mapsto x \geq 2 \wedge y \geq 2, b_3 \mapsto x \geq 2\}$ . The property  $\psi = \mathsf{X} K$  is captured by the NFA  $\rightarrow(\psi) \xrightarrow{K} (\top)$ . The product automata  $\mathcal{N}_{\mathcal{B},b_1}^\psi$  and  $\mathcal{N}_{\mathcal{B},b_2}^\psi$  are as follows:



where the shaded nodes are final. The formulas in nodes were obtained by applying quantifier elimination to the formulas built using *update* according to Definition 10.  $\mathcal{N}_{\mathcal{B},b_3}^\psi$  consists only of the dummy transition and has no final states.

Definition 10 need not terminate if infinitely many non-equivalent formulas occur in the construction. In Sect. 4 we will identify a criterion that guarantees termination. First, we state the key correctness property, which lifts [28, Theorem 4.7] to LTL with configuration maps. Its proof is similar to the respective result in [28], and can be found in [27].

**Theorem 1.** *Let  $\psi \in \text{LTL}_f^{\mathcal{B}}$  and  $b \in B$  such that there is a finite product automaton  $\mathcal{N}_{\mathcal{B},b}^{\psi}$ . Then there is a final run  $\rho: (b, \alpha_0) \rightarrow^* (b_F, \alpha_F)$  of  $\mathcal{B}$  such that  $\rho \models_{\mathcal{K}} \psi$ , iff  $\mathcal{N}_{\mathcal{B},b}^{\psi}$  has a final state  $(b_F, q_F, \varphi)$  for some  $q_F$  and  $\varphi$  such that  $\varphi$  is satisfied by assignment  $\gamma$  with  $\gamma(\overline{V}_0) = \alpha_0(\overline{V})$  and  $\gamma(\overline{V}) = \alpha_F(\overline{V})$ .*

Thus, witnesses for  $\psi$  correspond to paths to final states in the product automaton: e.g., in  $\mathcal{N}_{\mathcal{B},b_1}^{\psi}$  in Example 3 the formula in the left final node is satisfied by  $\gamma(x_0) = \gamma(x) = \gamma(y) = 3$  and  $\gamma(y_0) = 0$ . For  $\alpha_0$  and  $\alpha_2$  such that  $\alpha_0(\overline{V}) = \gamma(\overline{V}_0) = \{x \mapsto 3, y \mapsto 0\}$  and  $\alpha_2(\overline{V}) = \gamma(\overline{V}) = \{x \mapsto 3, y \mapsto 3\}$  there is a witness run for  $\psi$  from  $(b_1, \alpha_0)$  to  $(b_1, \alpha_2)$ , e.g.,  $(b_1, \left[ \begin{smallmatrix} x=3 \\ y=0 \end{smallmatrix} \right]) \xrightarrow{a_1} (b_2, \left[ \begin{smallmatrix} x=3 \\ y=3 \end{smallmatrix} \right]) \xrightarrow{a_3} (b_3, \left[ \begin{smallmatrix} x=3 \\ y=3 \end{smallmatrix} \right])$ .

## 4 Model Checking Procedure

Using the results of Sect. 3, we define a model checking procedure, shown in Fig. 1. First, we explain the tasks achieved by the three mutually recursive functions:

- *checkState*( $\chi$ ) returns a configuration map representing the set of configurations that satisfy a state formula  $\chi$ . In the base cases, it returns a function that checks the respective condition, for boolean operators we recurse on the arguments, and for a formula  $E\psi$  we proceed to the *checkPath* procedure.
- *checkPath*( $\psi$ ) returns a configuration map  $K$  that represents all configurations from which a path satisfying  $\psi$  exists. First, *toLTL $_{\mathcal{K}}$*  is used to obtain an equivalent  $\text{LTL}_f^{\mathcal{B}}$  formula  $\psi'$  (which entails the computation of solutions for all subproperties  $E\eta$ ). Then solution  $K$  is constructed as follows: For every control state  $b$ , we build the product automaton  $\mathcal{N}_{\mathcal{B},b}^{\psi'}$ , and collect the set  $\Phi_F$  of formulas in final states. Every  $\varphi \in \Phi_F$  encodes runs from  $b$  to a final state of  $\mathcal{B}$  that satisfy  $\psi'$ . The variables  $\overline{V}_0$  and  $\overline{V}$  in  $\varphi$  act as placeholders for the initial and the final values of the runs, respectively. We rename variables in  $\varphi$  to use  $\overline{V}$  at the start and  $\overline{U}$  at the end, we quantify existentially over  $\overline{U}$  (as the final valuation is irrelevant), and take the disjunction over all  $\varphi \in \Phi_F$ . The resulting formula  $\varphi'$  encodes all final runs from  $b$  that satisfy  $\psi'$ , so we set  $K(b) := \varphi'$ .
- *toLTL $_{\mathcal{K}}$* ( $\psi$ ) computes an  $\text{LTL}_f^{\mathcal{B}}$  formula equivalent to a path formula  $\psi$ . To this end, it performs two kinds of replacements in  $\psi$ : (a)  $\top$ ,  $b \in B$ , and constraints  $c$  are represented as configuration maps; and (b) subformulas  $E\eta$  are replaced by their solutions  $K_{E\eta}$ , which are computed by a recursive call to *checkPath*.

To represent the base cases of formulas as configuration maps in Fig. 1, we define  $K_{\top} := (\lambda \_. \top)$ ,  $K_b := (\lambda b'. b = b' ? \top : \perp)$  for all  $b \in B$ , and  $K_c := (\lambda \_. c)$  for constraints  $c$ . We also write  $\neg K$  for  $(\lambda b. \neg K(b))$  and  $K \wedge K'$  for  $(\lambda b. K(b) \wedge K'(b))$ . The next example illustrates the approach.

```

1: procedure checkState( $\chi$ )
2:   switch  $\chi$  do
3:     case  $\top$ ,  $b \in B$ , or  $c \in C$ : return  $K_\chi$ 
4:     case  $\chi_1 \wedge \chi_2$ :       return checkState( $\chi_1$ )  $\wedge$  checkState( $\chi_2$ )
5:     case  $\neg\chi$ :             return  $\neg$ checkState( $\chi$ )
6:     case  $E\psi$ :             return checkPath( $\psi$ )

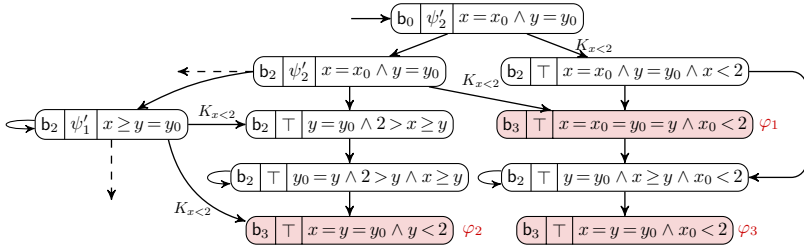
1: procedure checkPath( $\psi$ )
2:    $\psi' := \text{toLTL}_K(\psi)$ 
3:   for  $b \in B$  do
4:      $(P, R, p_0, P_F) := \mathcal{N}_{\mathcal{B}, b}^{\psi'}$   $\triangleright$  product automaton for  $\psi'$ ,  $\mathcal{B}$ , and  $b$ 
5:      $\Phi := \{\varphi \mid (b_F, q_F, \varphi) \in P_F\}$   $\triangleright$  collect formulas in final states
6:      $K(b) := \bigvee_{\varphi \in \Phi} \exists \bar{U}. \varphi(\bar{V}, \bar{U})$ 
7:   return  $K$ 

1: procedure toLTL $_K(\psi)$ 
2:   switch  $\psi$  do
3:     case  $\top$ ,  $b \in B$ , or  $c \in C$ : return  $K_\psi$ 
4:     case  $\psi_1 \wedge \psi_2$ :       return toLTL $_K(\psi_1) \wedge$  toLTL $_K(\psi_2)$ 
5:     case  $\neg\psi$ :             return  $\neg$ toLTL $_K(\psi)$ 
6:     case  $E\psi$ :             return checkPath( $\psi$ )
7:     case  $X\psi$ :             return  $X$  toLTL $_K(\psi)$ 
8:     case  $G\psi$ :             return  $G$  toLTL $_K(\psi)$ 
9:     case  $\psi_1 U \psi_2$ :     return toLTL $_K(\psi_1) U$  toLTL $_K(\psi_2)$ 

```

Fig. 1. Model checking procedure.

*Example 4.* Consider  $\chi = EX(AG(x \geq 2))$  and the DDSA  $\mathcal{B}$  in Example 2. To get a solution  $K_1$  to *checkState*( $\chi$ ) = *checkPath*( $\psi_1$ ) for  $\psi_1 = X(AG(x \geq 2))$ , we first compute an equivalent  $LTL_f^B$  formula  $\psi'_1 = XK_2$ , where  $K_2$  is a solution to  $AG(x \geq 2) \equiv \neg EF(x < 2)$ . To this end, we run *checkPath*( $\psi_2$ ) for  $\psi_2 = F(x < 2)$ , which is represented in  $LTL_f^B$  as  $\psi'_2 = F(K_{x < 2})$  with NFA  $\mathcal{A}^{\psi'_2} \xrightarrow{K_{x < 2}} \textcircled{\top}$ . Next, *checkPath* builds  $\mathcal{N}_{\mathcal{B}, b}^{\psi'_2}$  for all states  $b$ . For instance, for  $b_2$  we get:



where dashed arrows indicate transitions to non-final sink states. For  $\bar{U} = \langle \hat{x}, \hat{y} \rangle$ , and the formulas  $\varphi_1$ ,  $\varphi_2$ , and  $\varphi_3$  in final nodes, we compute

$$\begin{aligned}
\exists \bar{U}. \varphi_1(\bar{V}, \bar{U}) &= \exists \hat{x} \hat{y}. \hat{x} = x = \hat{y} = y \wedge x < 2 \equiv x < 2 \\
\exists \bar{U}. \varphi_2(\bar{V}, \bar{U}) &= \exists \hat{x} \hat{y}. \hat{x} = \hat{y} = y \wedge \hat{y} < 2 \equiv y < 2 \\
\exists \bar{U}. \varphi_3(\bar{V}, \bar{U}) &= \exists \hat{x} \hat{y}. \hat{x} = \hat{y} = y \wedge x < 2 \equiv x < 2
\end{aligned}$$

so that  $K_3 := \text{checkPath}(\psi_2)$  sets  $K_3(b_2) = \bigvee_{i=1}^3 \exists \bar{U}. \varphi_i(\bar{V}, \bar{U}) \equiv x < 2 \vee y < 2$ . For reasons of space, the constructions for  $b_1$  and  $b_3$  are shown in [27, Appendix B]; we obtain  $K_3(b_1) = \top$  and  $K_3(b_3) = x < 2$ . By negation, the solution  $K_2$  to  $\text{AG}(x \geq 2)$  is  $K_2 = \neg K_3 = \{b_1 \mapsto \perp, b_2 \mapsto x \geq 2 \wedge y \geq 2, b_3 \mapsto x \geq 2\}$ . Now we can proceed with  $\text{checkPath}(\psi_1)$ . The NFA and product automata for  $\psi'_1 = \text{X} K_2$  are as shown in Example 3 and in a similar way as above we obtain the solution  $K_1$  for  $\text{EXAG}(x \geq 2)$  as  $K_1 = \{b_1 \mapsto x \geq 2, b_2 \mapsto y \geq 2, b_3 \mapsto \perp\}$ . Thus,  $\mathcal{B}$  satisfies the property for any initial assignment  $\alpha_I$  with  $\alpha_I(x) \geq 2$ .

Next we prove correctness of  $\text{checkState}(\chi)$  under the condition that it is defined, i.e., all required product automata are finite. First we state our main result, but before giving its proof we show helpful properties of  $\text{toLTL}_{\mathcal{K}}$  and  $\text{checkPath}$ .

**Theorem 2.** *For every configuration  $(b, \alpha)$  of the DDSA  $\mathcal{B}$  and every state property  $\chi$ , if  $\text{checkState}(\chi)$  is defined then  $(b, \alpha) \models \chi$  iff  $\alpha \models \text{checkState}(\chi)(b)$ .*

**Lemma 3.** *Let  $\psi$  be a path formula with  $qd(\psi) = k$ . Suppose that for all configurations  $(b, \alpha)$  and path formulas  $\psi'$  with  $qd(\psi') < k$ , there is a  $\rho' \in \text{FRuns}(b, \alpha)$  with  $\rho' \models \psi'$  iff  $\alpha \models \text{checkPath}(\psi')(b)$ . Then  $\rho \models \psi$  iff  $\rho \models_{\mathcal{K}} \text{toLTL}_{\mathcal{K}}(\psi)$ .*

*Proof (sketch).* By induction on  $\psi$ . The base cases are by the definitions of  $K_{\top}$ ,  $K_b$ , and  $K_c$ . In the induction step, if  $\psi = \text{E} \psi'$  then  $\rho \models \psi$  iff  $\exists \rho' \in \text{FRuns}(b_0, \alpha_0)$  with  $\rho' \models \psi'$ , for  $\rho_0 = (b_0, \alpha_0)$ . As  $qd(\psi') < qd(\psi)$ , this holds by assumption iff  $\alpha_0 \models \text{checkPath}(\psi')(b_0)$ . This is equivalent to  $\rho \models_{\mathcal{K}} \text{toLTL}_{\mathcal{K}}(\psi) = \text{checkPath}(\psi')$ . All other cases are by the induction hypothesis and Definitions 7 and 9.

**Lemma 4.** *If  $\psi' = \text{toLTL}_{\mathcal{K}}(\psi)$  such that for all runs  $\rho$  it is  $\rho \models \psi$  iff  $\rho \models_{\mathcal{K}} \psi'$ , there is a run  $\rho \in \text{FRuns}(b, \alpha)$  with  $\rho \models \psi$  iff  $\alpha \models \text{checkPath}(\psi)(b)$ .*

*Proof.* ( $\Rightarrow$ ) Suppose there is a run  $\rho \in \text{FRuns}(b, \alpha)$  with  $\rho \models \psi$ , so  $\rho$  is of the form  $(b, \alpha) \rightarrow^* (b_F, \alpha_F)$  for some  $b_F \in B_F$ . By assumption, this implies  $\rho \models_{\mathcal{K}} \psi'$ , so that by Theorem 1,  $\mathcal{N}_{\mathcal{B}, b}^{\psi'}$  has a final state  $(b_F, q_F, \varphi)$  where  $\varphi$  is satisfied by an assignment  $\gamma$  with domain  $V \cup V_0$  such that  $\gamma(\bar{V}_0) = \alpha(\bar{V})$  and  $\gamma(\bar{V}) = \alpha_F(\bar{V})$ . By definition,  $\text{checkPath}(\psi)(b)$  contains a disjunct  $\exists \bar{U}. \varphi(\bar{V}, \bar{U})$ . As  $\gamma$  satisfies  $\varphi$  and  $\gamma(\bar{V}_0) = \alpha(\bar{V})$ ,  $\alpha \models \text{checkPath}(\psi)(b)$ . ( $\Leftarrow$ ) If  $\alpha \models \text{checkPath}(\psi)(b)$ , by definition of  $\text{checkPath}$  there is a formula  $\varphi$  such that  $\alpha \models \exists \bar{U}. \varphi(\bar{V}, \bar{U})$  and  $\varphi$  occurs in a final state  $(b_F, q_F, \varphi)$  of  $\mathcal{N}_{\mathcal{B}, b}^{\psi'}$ . Hence there is an assignment  $\gamma$  with domain  $V \cup V_0$  and  $\gamma(\bar{V}_0) = \alpha(\bar{V})$  such that  $\gamma \models \varphi$ . By Theorem 1, there is a run  $\rho: (b, \alpha) \rightarrow^* (b_F, \alpha_F)$  such that  $\rho \models_{\mathcal{K}} \psi'$ . By the assumption, we have  $\rho \models \psi$ .  $\square$

At this point the main theorem can be proven:

*Proof (of Theorem 2).* We first show  $(\star)$ : for any path formula  $\psi$ , there is a run  $\rho \in \text{FRuns}(b, \alpha)$  with  $\rho \models \psi$  iff  $\alpha \models \text{checkPath}(\psi)(b)$ . The proof is by induction on  $qd(\psi)$ . If  $\psi$  contains no path quantifiers, Lemma 3 implies that  $\rho \models \psi$  iff  $\rho \models_{\mathcal{K}} \text{toLTL}_{\mathcal{K}}(\psi)$  for all runs  $\rho$ , so  $(\star)$  follows from Lemma 4. In the induction step, we conclude from Lemma 3, using the induction hypothesis of

( $\star$ ) as assumption, that  $\rho \models \psi$  iff  $\rho \models_{\mathcal{K}} \text{toLTL}_{\mathcal{K}}(\psi)$  for all runs  $\rho$ . Again, ( $\star$ ) follows from Lemma 4.

The theorem is then shown by induction on  $\chi$ : The base cases  $\top$ ,  $b' \in B$ ,  $c \in \mathcal{C}$  are easy to check, and for properties of the form  $\neg\chi'$  and  $\chi_1 \wedge \chi_2$  the claim follows from the induction hypothesis and the definitions. Finally, for  $\chi = \mathbf{E}\psi$ ,  $(b, \alpha) \models \chi$  iff there is a run  $\rho \in \text{FRuns}(b, \alpha)$  such that  $\rho \models \psi$ . By ( $\star$ ) this is the case iff  $\alpha \models \text{checkPath}(\psi)(b) = \text{checkState}(\chi)(b)$ .  $\square$

*Termination.* We next show that the formulas generated in our procedure all have a particular shape, to obtain an abstract termination result. For a set of formulas  $\Phi \subseteq \mathcal{C}(V)$  and a symbolic run  $\sigma$ , let a history constraint  $h(\sigma, \bar{\vartheta})$  be *over basis*  $\Phi$  if  $\bar{\vartheta} = \langle \vartheta_0, \dots, \vartheta_n \rangle$  and for all  $i$ ,  $1 \leq i \leq n$ , there is a subset  $T_i \subseteq \Phi$  s.t.  $\vartheta_i = \bigwedge T_i$ . Moreover, for a set of formulas  $\Phi$ , let  $\Phi^\pm = \Phi \cup \{\neg\varphi \mid \varphi \in \Phi\}$ .

**Definition 11.** For a DDSA  $\mathcal{B}$ , a constraint set  $\mathcal{C}$  over free variables  $V$ , and  $k \geq 0$ , the formula sets  $\Phi_k$  are inductively defined by  $\Phi_0 = \mathcal{C} \cup \{\top, \perp\}$  and

$$\Phi_{k+1} = \left\{ \bigvee_{\varphi \in H} \exists \bar{U}. \varphi(\bar{V}, \bar{U}) \mid H \subseteq \mathcal{H}_k \right\}$$

where  $\mathcal{H}_k$  is the set of all history constraints of  $\mathcal{B}$  with basis  $\bigcup_{i \leq k} \Phi_i^\pm$ .

Note that formulas in  $\Phi_k$  have free variables  $V$ , while those in  $\mathcal{H}_k$  have free variables  $V_0 \cup V$ . We next show that these sets correspond to the formulas generated by our procedure, if all constraints in the verification property are in  $\mathcal{C}$ .

**Lemma 5.** Let  $\mathbf{E}\psi$  have quantifier depth  $k$ ,  $\psi' = \text{toLTL}_{\mathcal{K}}(\psi)$ , and  $\mathcal{N}_{\mathcal{B}, b}^{\psi'}$  be a constraint graph constructed in  $\text{checkPath}(\psi)$  for some  $b \in B$ . Then,

- (1) for all nodes  $(b', q, \varphi)$  in  $\mathcal{N}_{\mathcal{B}, b}^{\psi'}$  there is some  $\varphi' \in \mathcal{H}_k$  such that  $\varphi \equiv \varphi'$ ,
- (2)  $\text{checkPath}(\psi)(b)$  is equivalent to a formula in  $\Phi_{k+1}$ .

The statements are proven by induction on  $k$ , using the results on the product construction ([27, Lemma 6]). From part (1) of this lemma and Theorem 2 we thus obtain an abstract criterion for decidability that will be useful in the next section:

**Corollary 1.** For a DDSA  $\mathcal{B}$  as above and a state formula  $\chi$ , if  $\mathcal{H}_j(b)$  is finite up to equivalence for all  $j < \text{qd}(\chi)$  and  $b \in B$ , a witness map can always be computed.

*Proof.* By the assumption about the sets  $\mathcal{H}_j(b)$  for  $j < \text{qd}(\chi)$ , all product automata constructions in recursive calls  $\text{checkPath}(\psi)$  of  $\text{checkState}(\chi)$  terminate if logical equivalence of formulas is checked eagerly. Thus  $\text{checkState}(\chi)$  is defined, and by Theorem 2 the result is a witness map.  $\square$

The property that all sets  $\mathcal{H}_j(b)$ ,  $j < \text{qd}(\chi)$ , are finite might not be decidable itself. However, in the next section we will show means to guarantee this property. Moreover, we remark that finiteness of all  $\mathcal{H}_j(b)$  implies a *finite history set*, a decidability criterion identified for the linear-time case [28, Definition 3.6]; but Example 5 below illustrates that the requirement on the  $\mathcal{H}_j(b)$ 's is strictly stronger.

## 5 Decidability of DDSA Classes

We here illustrate restrictions on DDSAs, either on the control flow or on the constraint language, that render our approach a decision procedure for  $\text{CTL}_f^*$ .

**Monotonicity constraints** (MCs) restrict constraints (Definition 1) as follows: MCs over variables  $V$  and domain  $D$  have the form  $p \odot q$  where  $p, q \in D \cup V$  and  $\odot$  is one of  $=, \neq, \leq, <, \geq$ , or  $>$ . The domain  $D$  may be  $\mathbb{R}$  or  $\mathbb{Q}$ . We call a boolean formula whose atoms are MCs an *MC formula*, a DDSA where all atoms in guards are MCs an *MC-DDSA*, and a  $\text{CTL}_f^*$  property whose constraint atoms are MCs an *MC property*. For instance,  $\mathcal{B}$  in Example 2 is an MC-DDSA.

We exploit a useful quantifier elimination property: If  $\varphi$  is an MC formula over a set of constants  $L$  and variables  $V \cup \{x\}$ , there is some  $\varphi' \equiv \exists x. \varphi$  such that  $\varphi'$  is a quantifier-free MC formula over  $V$  and  $L$ . Such a  $\varphi'$  can be obtained by writing  $\varphi$  in disjunctive normal form and applying a Fourier-Motzkin procedure [36, Sect. 5.4] to each disjunct, which guarantees that all constants in  $\varphi'$  also occur in  $\varphi$ .

**Theorem 3.** *For any DDSA  $\mathcal{B}$  and property  $\chi$  over monotonicity constraints, a witness map is computable.*

*Proof.* Let  $\chi$  be an MC property, and  $L$  the finite set of constants in constraints in  $\chi$ ,  $\alpha_0$ , and guards of  $\mathcal{B}$ . Let moreover  $\text{MC}_L$  be the set of quantifier-free formulas whose atoms are MCs over  $V \cup V_0$  and  $L$ , so  $\text{MC}_L$  is finite up to equivalence.

We show the following property ( $\star$ ): all history constraints  $h(\sigma, \bar{\vartheta})$  over basis  $\text{MC}_L$  are equivalent to a formula in  $\text{MC}_L$ . For a symbolic run  $\sigma: b_0 \rightarrow^* b_{n-1} \xrightarrow{a} b_n$  and a sequence  $\bar{\vartheta} = \langle \vartheta_0, \dots, \vartheta_n \rangle$  over  $\text{MC}_L$ , the proof is by induction on  $n$ . In the base case,  $h(\sigma, \bar{\vartheta}) = \varphi_\nu \wedge \vartheta_0$  is in  $\text{MC}_L$  because  $\varphi_\nu$  is a conjunction of equalities between  $V \cup V_0$ , and  $\vartheta_0 \in \text{MC}_L$  by assumption. In the induction step,  $h(\sigma, \bar{\vartheta}) = \text{update}(h(\sigma|_{n-1}, \bar{\vartheta}|_{n-1}), a_n) \wedge \vartheta_n$ . By induction hypothesis,  $h(\sigma|_{n-1}, \bar{\vartheta}|_{n-1}) \equiv \varphi$  for some  $\varphi$  in  $\text{MC}_L$ . Thus  $h(\sigma, \bar{\vartheta}) \equiv \exists \bar{U}. \varphi(\bar{U}) \wedge \Delta_a(\bar{U}, \bar{V}) \wedge \vartheta_n$ . As  $\mathcal{B}$  is an MC-DDSA,  $\Delta_a(\bar{U}, \bar{V})$  is a conjunction of MCs over  $V \cup U$  and constants  $L$ , and  $\vartheta_n \in \text{MC}_L$  by assumption. By the quantifier elimination property, there exists a quantifier-free MC-formula  $\varphi'$  over variables  $V_0 \cup V$  that is equivalent to  $\exists \bar{U}. \varphi(\bar{U}) \wedge \Delta_a(\bar{U}, \bar{V}) \wedge \vartheta_n$ , and mentions only constants in  $L$ , so  $\varphi' \in \text{MC}_L$ .

For  $\mathcal{C}$  the set of constraints in  $\chi$ , we now show that  $\mathcal{H}_j \subseteq \text{MC}_L$  for all  $j \geq 0$ , by induction on  $j$ . In the base case ( $j = 0$ ), the claim follows from ( $\star$ ), as all constraints in  $\Phi_0$ , i.e., in  $\chi$ , are in  $\text{MC}_L$ . For  $j > 0$ , consider first a formula  $\hat{\varphi} \in \Phi_j$  for some  $b \in B$ . Then  $\hat{\varphi}$  is of the form  $\hat{\varphi} = \bigvee_{\varphi \in H} \exists \bar{U}. \varphi(\bar{V}, \bar{U})$  for some  $H \subseteq \mathcal{H}_{j-1}$ . By the induction hypothesis,  $H \subseteq \text{MC}_L$ , so by the quantifier elimination property of MC formulas,  $\hat{\varphi}$  is equivalent to an MC-formula over  $V$  and  $L$  in  $\text{MC}_L$ . As  $\mathcal{H}_j$  is built over basis  $\Phi_j$ , the claim follows from ( $\star$ ).  $\square$

Notably, the above quantifier elimination property fails for MCs over integer variables; indeed, CTL model checking is undecidable in this case [42, Theorem 4.1].



**Integer periodicity constraint** systems confine the constraint language to variable-to-constant comparisons and restricted forms of variable-to-variable comparisons, and are for instance used in calendar formalisms [18, 22]. More precisely, *integer periodicity constraint* (IPC) atoms have the form  $x = y$ ,  $x \odot d$  for  $\odot \in \{=, \neq, <, >\}$ ,  $x \equiv_k y + d$ , or  $x \equiv_k d$ , for variables  $x, y$  with domain  $\mathbb{Z}$  and  $k, d \in \mathbb{N}$ . A boolean formula whose atoms are IPCs is an *IPC formula*, a DDSA whose guards are conjunctions of IPCs an *IPC-DDSA*, and a  $\text{CTL}_f^*$  formula whose constraint atoms are IPCs an *IPC property*. For instance,  $\mathcal{B}_{ipc}$  in Example 2 is an IPC-DDSA.

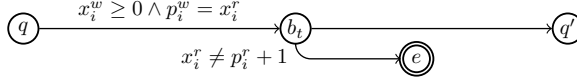
Using Corollary 1 and a known quantifier elimination property for IPCs [18, Theorem 2], one can show that witness maps are also computable for IPC-DDSAs, in a proof that resembles the one of Theorem 3 (see [27, Theorem 4]).

**Theorem 4.** *For any DDSA  $\mathcal{B}$  and property  $\chi$  over integer periodicity constraints, a witness map is computable.*

The proofs of both Theorems 3 and 4 rely on the fact that all transition guards and constraints in the verification property are in a *finite* set of constraints  $C$  that is closed under quantifier elimination, so that for all  $\varphi \in C$  and actions  $a$ ,  $\text{update}(\varphi, a)$  is again equivalent to a formula in  $C$ . However, this is not the only way to ensure the requirements of Corollary 1: For a simple example, these requirements are satisfied by a loop-free DDSA, where the number of runs is finite. Interestingly, while the cases of MC and IPC systems are also captured by the abstract decidability criterion by Gascon [31], this need not apply to loop-free DDSAs. A clarification of the relationship between the criteria in Corollary 1 and [31, Thm 4.5] requires further investigation.

**Bounded lookback** [28] restricts the control flow of a DDSA rather than the constraint language, and is a generalization of the earlier *feedback-freedom* property [14]. Intuitively,  $k$ -bounded lookback demands that the behavior of a DDSA at any point in time depends only on  $k$  events from the past. We refer to [28, Definition 5.9] for the formal definition. Systems that enjoy bounded lookback allow for decidable linear-time verification [28, Theorem 5.10]. However, we next show that this result does not extend to branching time.

*Example 5.* We reduce control state reachability of two-counter machines (2CM) to the verification problem of  $\text{CTL}_f^*$  formulas in bounded lookback systems, inspired by [42, Theorem 4.1]. 2CMs have a finite control structure and two counters  $x_1$ , and  $x_2$  that can be incremented, decremented, and tested for 0. It is undecidable whether a 2CM will ever reach a designated control state  $f$  [43]. For a 2CM  $\mathcal{M}$ , we build a feedback-free DDSA  $\mathcal{B} = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, \text{guard} \rangle$  and a  $\text{CTL}_f^*$  property  $\chi$  such that  $\mathcal{B}$  satisfies  $\chi$  iff  $f$  is reachable in  $\mathcal{M}$ . The set  $B$  consists of the control states of  $\mathcal{M}$ , together with an error state  $e$  and auxiliary states  $b_t$  for transitions  $t$  of  $\mathcal{M}$ , and  $B_F = \{f, e\}$ . The set  $V$  consists of  $x_1$ ,  $x_2$  and auxiliary variables  $p_1$ ,  $p_2$ ,  $m_1$ ,  $m_2$ . Zero-test transitions of  $\mathcal{M}$  are directly modeled in  $\mathcal{B}$ , whereas a step  $q \rightarrow q'$  that increments  $x_i$  by one is modeled as:



The step  $q \rightarrow b_t$  writes  $x_i$ , storing its previous value in  $p_i$ , but if the write was not an increment by exactly 1, a step to state  $e$  is enabled. Decrements are modeled similarly. Intuitively, bounded lookback holds because variable dependencies are limited: in a run of  $\mathcal{M}$ , a variable dependency that is not an equality extends over at most two time points. (More formally, non-equality paths in the computation graph have at most length 1.) As increments are not exact,  $\mathcal{B}$  overapproximates  $\mathcal{M}$ . However,  $\chi = \text{EG}(\neg \text{EX} e)$  asserts existence of a path that never allows for a step to  $e$  (i.e., it properly simulates  $\mathcal{M}$ ) but reaches the final state  $f$ . Thus,  $\mathcal{B}$  satisfies  $\chi$  iff  $f$  is reachable in  $\mathcal{M}$ .

## 6 Implementation

We implemented our approach in the prototype **ada** (arithmetic DDS analyzer) in Python; source code, benchmarks, and a web interface are available (<https://ctlstar.adatool.dev>). As input, the tool takes a CTL\* property  $\chi$  together with a DDSA in JSON format; alternatively, a given (bounded) Petri net with data (DPN) in PNML format [5] can be transformed into a DDSA. The tool then applies the algorithm in Fig. 1. If successful, it outputs the configuration map returned by  $\text{checkState}(\chi)$ , and it can visualize the product constructions. For SMT checks and quantifier elimination, **ada** interfaces CVC5 [23] and Z3 [17]. Besides numeric variables, **ada** also supports variables of type boolean and string; for the latter, only equality comparison is supported, so different constants can be represented by distinct integers. In addition to the operations in Definition 6, **ada** allows next operators  $\langle a \rangle$  via an action  $a$ , which are useful for verification.

We tested **ada** on a set of business process models presented as Data Petri nets (DPNs) in the literature. As these nets are bounded, they can be transformed into DDSAs. The results are reported in the table below. We indicate whether the system belongs to a decidable class, the verified property and whether it is satisfied by the initial configuration, the verification time, the number of SMT checks, and the number of nodes in the DDSA  $\mathcal{B}$  and the sum of all product constructions, respectively. We used CVC5 as SMT solver; times are without visualization, which tends to be time-consuming for large graphs. All tests were run on an Intel Core i7 with 4×2.60 GHz and 19 GB RAM.

process	property	sat	time	checks	$ \mathcal{B} $	$ \mathcal{N}_{\mathcal{B},b}^\psi $
(a) road fines	No deadlock	✗	7.0s	8161	9	2052
	$\text{AG } (p_7 \rightarrow \text{E F end})$	✓	7.6s	7655		1987
	$\text{AG } (\text{end} \rightarrow \text{total} \leq \text{amount})$	✗	1m12s	111139		3622
(b) road fines	No deadlock	✓	15m27s	247563	9	4927
	$\text{AG } (p_7 \rightarrow \text{E F end})$	✓	16m7s	246813		4927
(c) road fines	No deadlock	✗	9s	9179	9	1985
	$\text{AG } (p_7 \rightarrow \text{E F end})$	✓	6.6s	6382		1597
	$\psi_{c1} = \text{EF } (dS \geq 2160)$	✗	11.5s	17680		1280
	$\psi_{c2} = \text{EF } (dP \geq 1440)$	✗	10.0s	15187		1280
	$\psi_{c3} = \text{EF } (dJ \geq 1440)$	✗	10.5	16000		1280
	No deadlock	✓	20m59s	1234928	17	23147
(d) hospital billing	$\psi_{d1} = \text{EF } (p16 \wedge \neg \text{closed})$	✓	10m20s	669379		10654
	No deadlock	✓	1m36s	139	301	44939
	$\psi_{e1} = \text{AG } (\text{sink} \rightarrow t_{tr} < t_{ab})$	✗	30.1s	170		22724
(e) sepsis	$\psi_{e2} = \text{AG } (\text{sink} \rightarrow t_{tr} + 60 \geq t_{ab})$	✓	32s	153		22538
	No deadlock	✓	7m24	4524	301	161242
	$\psi_{f1} = \text{A } (\neg \text{lacticAcidG } \langle \text{diagnostic} \rangle \top)$	✓	3m53s	5734		74984
(g) board: register	No deadlock	✓	1.4s	12	7	27
(h) board: transfer	No deadlock	✓	1.4s	27	7	51
(i) board: discharge	No deadlock	✓	1.5s	25	6	67
	$\psi_{i1} = \text{AG } (p_2 \wedge o_1 = 207 \rightarrow \text{AG } o_1 = 207)$	✓	1.5s	94		91
	$\psi_{i2} = \text{A } (\text{EF } \langle \text{tra} \rangle \top \wedge \text{EF } \langle \text{his} \rangle \top)$	✓	1.5s	27		98
	$\psi_{i3} = \neg \text{E } (\text{F } \langle \text{tra} \rangle \top \wedge \text{F } \langle \text{his} \rangle \top)$	✓	1.4s	56		43
(j) credit approval	No deadlock	✓	1.7s	470	6	230
	$\psi_{j1} = \text{AG } ((\text{openLoan}) \top \rightarrow \text{ver} \wedge \text{dec})$	✓	13.2s	14156		645
	$\psi_{j2} = \text{A } (\text{F } (\text{ver} \wedge \text{dec}) \rightarrow \text{F } \langle \text{openLoan} \rangle \top)$	✗	3.7s	3128		316
	$\psi_{j3} = \text{A } (\text{F } (\text{ver} \wedge \text{dec}) \rightarrow \text{EF } \langle \text{openLoan} \rangle \top)$	✓	5.6s	4748		548
(k) package handling	No deadlock	✓	2.7ss	1025	16	693
	No deadlock ( $\tau_1$ )	✓	2.5s	1079		398
	$\psi_{k1} = \text{EF } \langle \text{fetch} \rangle \top$	✗	2.6s	850		343
	$\psi_{k2} = \text{EF } \langle \tau_6 \rangle \top$	✗	2.4s	875		336
(l) auction	No deadlock	✗	10.8s	1683	5	186
	$\text{EF } (\text{sold} \wedge d > 0 \wedge o \leq t)$	✗	6.4s	1180		79
	$\text{EF } (b = 1 \wedge o > t \wedge \text{F } (\text{sold} \wedge b > 1))$	✓	26.5s	4000		263

We briefly comment on the benchmarks and some properties: For all examples we checked *no deadlock*, which abbreviates  $\text{AG EF } \chi_f$  where  $\chi_f$  is a disjunction of all final states. This is one of the two requirements of the crucial *soundness* property (cf. Example 1). Weak soundness [4] relaxes this requirement to demand only that if a transition is reachable, it does not lead to deadlocks; this is called here *no deadlock(a)*, expressed by  $\text{EF } (\langle a \rangle \top) \rightarrow \text{AG } (\langle a \rangle \top \rightarrow \text{F } \chi_f)$ . One can also check whether a specific state  $p$  is deadlock-free, via  $\text{AG } (p \rightarrow \text{EF } \chi_f)$ .

- (a)–(c) are versions of the road fine management process (cf. Example 1); (a) [40, Fig. 12.7] and (b) [37, Fig. 13] were mined automatically from logs, while (c) is the normative version [41, Fig. 7] shown in Example 1. While in (a) and (c) *no deadlock* is violated, this issue was fixed in version (b). The fact that  $\psi_{c1}$ ,  $\psi_{c2}$ , and  $\psi_{c3}$  hold confirm that the time constraints are never violated.
- (d) models a billing process in a hospital [40, Fig. 15.3], which is deadlock-free.
- (e) is a normative model for a sepsis triage process in a hospital [40, Fig. 13.3], and (f) is a variation that was mined purely automatically from logs [40, Fig. 13.6]. According to [40, Sect. 13], triage should happen before antibiotics are administered, expressed by  $\psi_{e1}$ , which is actually not satisfied. However, the desired time constraint expressed by  $\psi_{e2}$  holds.
- (g)–(i) reflect activities in patient logistics of a hospital, based on logs of real-life processes [40, Fig. 14.3]. While the *no deadlock* property is satisfied by

all initial configurations, the output of **ada** reveals that for (h) this need not hold for other initial assignments.

- (j) is a credit approval process [16, Fig. 3]. It is deadlock-free;  $\psi_{j1}$  and  $\psi_{j2}$  verify desirable conditions under which a loan is granted to a client.
- (k) is a package handling routine [26, Fig. 5]. The fact that the properties  $\psi_{k1}$  and  $\psi_{k2}$  are not satisfied shows that the transitions  $\tau_6$  and **fetch** are dead.
- (l) models an auction process [28, Example 1.1], for which **ada** reveals a deadlock. Results for two further properties from [28, Example 1.1] are listed as well.

Seven systems are in a decidable class wrt. the listed properties: (a), (b), (d), (f), (h), (i), (k) are MC, while (d), (h), (i), (k) are IPC. This is due to the fact that automatic mining techniques often produce monotonicity constraints [39].

## 7 Conclusion

This paper presents a technique to compute witness maps for a given DDSA and  $\text{CTL}_f^*$  property, where a witness map specifies conditions on the initial variable assignment such that the property holds. The addressed problem is thus a slight generalization of the common verification problem. While our model checking procedure need not terminate in general, we show that it does if an abstract property on history constraints holds. Moreover, witness maps always exist for monotonicity and integer periodicity constraint systems. However, this result does not extend to bounded lookback systems. We implemented our approach in the tool **ada** and showed its usefulness on a range of business process models.

We see various opportunities to extend this work. A richer verification language could support past time operators [18] and future variable values [20]. Further decidable fragments could be sought using covers [33], or aiming for compatibility with locally finite theories [32]. Moreover, a restricted version of the bounded lookback property could guarantee decidability of  $\text{CTL}_f^*$ , similarly to the way feedback freedom was strengthened in [35]. The implementation could be improved to avoid the computation of many similar formulas, thus gaining efficiency. Finally, the complexity class that our approach implies for  $\text{CTL}_f^*$  in the decidable classes is yet to be clarified.

## References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016). <https://doi.org/10.1007/978-3-662-49851-4>
2. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
3. Baral, C., De Giacomo, G.: Knowledge representation and reasoning: what's hot. In: Proceedings of the 29th AAAI, pp. 4316–4317 (2015)
4. Batoulis, K., Haarmann, S., Weske, M.: Various notions of soundness for decision-aware business processes. In: Mayr, H.C., Guizzardi, G., Ma, H., Pastor, O. (eds.) ER 2017. LNCS, vol. 10650, pp. 403–418. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-69904-2\\_31](https://doi.org/10.1007/978-3-319-69904-2_31)

5. Billington, J., et al.: The petri net markup language: concepts, technology, and tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-44919-1\\_31](https://doi.org/10.1007/3-540-44919-1_31)
6. Bozga, M., Gîrlea, C., Iosif, R.: Iterating octagons. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 337–351. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-00768-2\\_29](https://doi.org/10.1007/978-3-642-00768-2_29)
7. Bozzelli, L., Gascon, R.: Branching-time temporal logic extended with qualitative Presburger constraints. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 197–211. Springer, Heidelberg (2006). [https://doi.org/10.1007/11916277\\_14](https://doi.org/10.1007/11916277_14)
8. Bozzelli, L., Pinchinat, S.: Verification of gap-order constraint abstractions of counter systems. *Theor. Comput. Sci.* **523**, 1–36 (2014). <https://doi.org/10.1016/j.tcs.2013.12.002>
9. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: a database theory perspective. In: Proceedings of the 32nd PODS, pp. 1–12 (2013). <https://doi.org/10.1145/2463664.2467796>
10. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: First-order  $\mu$ -calculus over generic transition systems and applications to the situation calculus. *Inf. Comput.* **259**(3), 328–347 (2018). <https://doi.org/10.1016/j.ic.2017.08.007>
11. Carapelle, C., Kartzow, A., Lohrey, M.: Satisfiability of ECTL\* with constraints. *J. Comput. Syst. Sci.* **82**(5), 826–855 (2016). <https://doi.org/10.1016/j.jcss.2016.02.002>
12. Čerāns, K.: Deciding properties of integral relational automata. In: Abiteboul, S., Shamir, E. (eds.) ICALP 1994. LNCS, vol. 820, pp. 35–46. Springer, Heidelberg (1994). [https://doi.org/10.1007/3-540-58201-0\\_56](https://doi.org/10.1007/3-540-58201-0_56)
13. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and Presburger arithmetic. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0028751>
14. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.* **37**(3), 22:1–22:36 (2012). <https://doi.org/10.1145/2338626.2338628>
15. de Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: insensitivity to infiniteness. In: Proceedings of the 28th AAAI, pp. 1027–1033 (2014)
16. de Leoni, M., Mannhardt, F.: Decision discovery in business processes. In: Encyclopedia of Big Data Technologies, pp. 1–12. Springer (2018). [https://doi.org/10.1007/978-3-319-63962-8\\_96-1](https://doi.org/10.1007/978-3-319-63962-8_96-1)
17. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_24](https://doi.org/10.1007/978-3-540-78800-3_24)
18. Demri, S.: LTL over integer periodicity constraints. *Theor. Comput. Sci.* **360**(1–3), 96–123 (2006). <https://doi.org/10.1016/j.tcs.2006.02.019>
19. Demri, S., Dhar, A.K., Sangnier, A.: Equivalence between model-checking flat counter systems and Presburger arithmetic. *Theor. Comput. Sci.* **735**, 2–23 (2018). <https://doi.org/10.1016/j.tcs.2017.07.007>
20. Demri, S., D’Souza, D.: An automata-theoretic approach to constraint LTL. *Inform. Comput.* **205**(3), 380–415 (2007). <https://doi.org/10.1016/j.ic.2006.09.006>
21. Demri, S., Finkel, A., Goranko, V., van Drimmelen, G.: Model-checking CTL\* over flat Presburger counter systems. *J. Appl. Non Class. Logics* **20**(4), 313–344 (2010). <https://doi.org/10.3166/jancl.20.313-344>

22. Demri, S., Gascon, R.: Verification of qualitative Z constraints. *Theor. Comput. Sci.* **409**(1), 24–40 (2008). <https://doi.org/10.1016/j.tcs.2008.07.023>
23. Deters, M., Reynolds, A., King, T., Barrett, C.W., Tinelli, C.: A tour of CVC4: how it works, and how to use it. In: *Proceedings of the 14th FMCAD*, p. 7 (2014). <https://doi.org/10.1109/FMCAD.2014.6987586>
24. Deutsch, A., Hull, R., Li, Y., Vianu, V.: Automatic verification of database-centric systems. *ACM SIGLOG News* **5**(2), 37–56 (2018). <https://doi.org/10.1145/3212019.3212025>
25. Felli, P., de Leoni, M., Montali, M.: Soundness verification of decision-aware process models with variable-to-variable conditions. In: *Proceedings of the 19th ACSD*, pp. 82–91. IEEE (2019). <https://doi.org/10.1109/ACSD.2019.00013>
26. Felli, P., de Leoni, M., Montali, M.: Soundness verification of data-aware process models with variable-to-variable conditions. *Fund. Inform.* **182**(1), 1–29 (2021). <https://doi.org/10.3233/FI-2021-2064>
27. Felli, P., Montali, M., Winkler, S.: CTL\* model checking for data-aware dynamic systems with arithmetic (extended version) (2022). <https://doi.org/10.48550/arXiv.2205.08976>
28. Felli, P., Montali, M., Winkler, S.: Linear-time verification of data-aware dynamic systems with arithmetic. In: *Proceedings of the 36th AAAI* (2022). <https://doi.org/10.48550/arXiv.2203.07982>
29. Finkel, A., Leroux, J.: How to compose Presburger-accelerations: applications to broadcast protocols. In: Agrawal, M., Seth, A. (eds.) *FSTTCS 2002*. LNCS, vol. 2556, pp. 145–156. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-36206-1\\_14](https://doi.org/10.1007/3-540-36206-1_14)
30. Finkel, A., Willems, B., Wolper, P.: A direct symbolic approach to model checking pushdown systems. In: *Proc. 2nd INFINITY*. ENTCS, vol. 9, pp. 27–37 (1997). [https://doi.org/10.1016/S1571-0661\(05\)80426-8](https://doi.org/10.1016/S1571-0661(05)80426-8)
31. Gascon, R.: An automata-based approach for CTL\* with constraints. In: *Proceedings of the INFINITY 2006, 2007 and 2008*. ENTCS, vol. 239, pp. 193–211 (2009). <https://doi.org/10.1016/j.entcs.2009.05.040>
32. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Combination methods for satisfiability and model-checking of infinite-state systems. In: Pfenning, F. (ed.) *CADE 2007*. LNCS (LNAI), vol. 4603, pp. 362–378. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73595-3\\_25](https://doi.org/10.1007/978-3-540-73595-3_25)
33. Gulwani, S., Musuvathi, M.: Cover algorithms and their combination. In: Drossopoulou, S. (ed.) *ESOP 2008*. LNCS, vol. 4960, pp. 193–207. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78739-6\\_16](https://doi.org/10.1007/978-3-540-78739-6_16)
34. Ibarra, O.H., Su, J.: Counter machines: decision problems and applications. In: *Jewels are Forever: Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pp. 84–96 (1999)
35. Koutsos, A., Vianu, V.: Process-centric views of data-driven business artifacts. *J. Comput. Syst. Sci.* **86**, 82–107 (2017). <https://doi.org/10.1016/j.jcss.2016.11.012>
36. Kroening, D., Strichman, O.: *Decision Procedures - An Algorithmic Point of View*. Second Edition. Springer (2016). <https://doi.org/10.1007/978-3-662-50497-0>
37. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) *ER 2018*. LNCS, vol. 11157, pp. 219–235. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-00847-5\\_17](https://doi.org/10.1007/978-3-030-00847-5_17)
38. de Leoni, M., Felli, P., Montali, M.: Strategy synthesis for data-aware dynamic systems with multiple actors. In: *Proceedings of the 17th KR*, pp. 315–325 (2020). <https://doi.org/10.24963/kr.2020/32>

39. de Leoni, M., Felli, P., Montali, M.: Integrating BPMN and DMN: modeling and analysis. *J. Data Semant.* **10**(1), 165–188 (2021). <https://doi.org/10.1007/s13740-021-00132-z>
40. Mannhardt, F.: Multi-perspective process mining. Ph.D. thesis, Technical University of Eindhoven (2018)
41. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. *Computing* **98**(4), 407–437 (2015). <https://doi.org/10.1007/s00607-015-0441-1>
42. Mayr, R., Totzke, P.: Branching-time model checking gap-order constraint systems. *Fundam. Informaticae* **143**(3–4), 339–353 (2016). <https://doi.org/10.3233/FI-2016-1317>
43. Minsky, M.: *Computation: Finite and Infinite Machines*. Prentice-Hall (1967)
44. Murano, A., Parente, M., Rubin, S., Sorrentino, L.: Model-checking graded computation-tree logic with finite path semantics. *Theor. Comput. Sci.* **806**, 577–586 (2020). <https://doi.org/10.1016/j.tcs.2019.09.021>
45. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: *Comptes Rendus du I congrès de Mathem. des Pays Slaves*, pp. 92–101 (1929)
46. Reichert, M.: Process and data: two sides of the same coin? In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F. (eds.) *OTM 2012. LNCS*, vol. 7565, pp. 2–19. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-33606-5\\_2](https://doi.org/10.1007/978-3-642-33606-5_2)
47. Sorrentino, L., Rubin, S., Murano, A.: Graded CTL\* over finite paths. In: *Proceedings of the 19th ICTCS. CEUR Workshop Proceedings*, vol. 2243, pp. 152–161. CEUR-WS.org (2018)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.




The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.







# SAT-Based Proof Search in Intermediate Propositional Logics

Camillo Fiorentini<sup>1</sup>  and Mauro Ferrari<sup>2</sup>  

<sup>1</sup> Department of Computer Science, Università degli Studi di Milano, Milan, Italy

<sup>2</sup> Department of Theoretical and Applied Sciences,  
Università degli Studi dell'Insubria, Varese, Italy  
`mauro.ferrari@uninsubria.it`

**Abstract.** We present a decision procedure for intermediate logics relying on a modular extension of the SAT-based prover `intuitR` for IPL (Intuitionistic Propositional Logic). Given an intermediate logic  $L$  and a formula  $\alpha$ , the procedure outputs either a Kripke countermodel for  $\alpha$  or the instances of the characteristic axioms of  $L$  that must be added to IPL in order to prove  $\alpha$ . The procedure exploits an incremental SAT-solver; during the computation, new clauses are learned and added to the solver.

## 1 Introduction

Recently, Claessen and Rosén have introduced `intuit` [4], an efficient decision procedure for Intuitionistic Propositional Logic (IPL) based on the Satisfiability Modulo Theories (SMT) approach. The prover language consists of (flat) clauses of the form  $\bigwedge A_1 \rightarrow \bigvee A_2$  (with  $A_i$  a set of atoms), which are fed to the SAT-solver, and implication clauses of the form  $(a \rightarrow b) \rightarrow c$  ( $a, b, c$  atoms); thus, we need an auxiliary clausification procedure to preprocess the input formula. The search is performed via a proper variant of the DPLL( $\mathcal{T}$ ) procedure [16], by exploiting an incremental SAT-solver; during the computation, whenever a semantic conflict is thrown, a new clause is learned and added to the SAT-solver. As discussed in [9], there is a close connection between the `intuit` approach and the known proof-theoretic methods. Actually, the decision procedure mimics the standard root-first proof search strategy for a sequent calculus strongly connected with Dyckhoff's calculus LJ<sub>T</sub> [5] (alias G4ip). To improve performances, we have re-designed the prover by adding a restart operation, thus obtaining `intuitR` [8] (`intuit` with Restart). Differently from `intuit`, the `intuitR` procedure has a simple structure, consisting of two nested loops. Given a formula  $\alpha$ , if  $\alpha$  is provable in IPL the call `intuitR( $\alpha$ )` yields a derivation of  $\alpha$  in the sequent calculus introduced in [8], a plain calculus where derivations have a single branch. If  $\alpha$  is not provable in IPL, the outcome of `intuitR( $\alpha$ )` is a (typically small) countermodel for  $\alpha$ , namely a Kripke model falsifying  $\alpha$ . We stress that `intuitR` is highly performant: on the basis of a standard benchmarks suite, it outperforms `intuit` and other state-of-the-art provers (in particular, `fCube` [6] and `intHistGC` [12]).



In this paper we present `intuitRIL`, an extension of `intuitR` to Intermediate Logics, namely propositional logics extending IPL and contained in CPL (Classical Propositional Logic). Specifically, let  $\alpha$  be a formula and  $L$  an axiomatizable intermediate logic having Kripke semantics; the call `intuitRIL( $\alpha, L$ )` tries to prove the validity of  $\alpha$  in  $L$ . To this aim, the prover searches for a set  $\Psi$  containing instances of  $\text{Ax}(L)$ , the characteristic axioms of  $L$ , such that  $\alpha$  can be proved in IPL from  $\Psi$ . Note that this is different from other approaches, where the focus is on the synthesis of specific inference rules for the logic at hand (see, e.g., [17]). Basically, `intuitRIL( $\alpha, L$ )` searches for a countermodel  $\mathcal{K}$  for  $\alpha$ , exploiting the search engine of `intuitR`: whenever we get  $\mathcal{K}$ , we check whether  $\mathcal{K}$  is a model of  $L$ . If this is the case, we conclude that  $\alpha$  is not valid in  $L$  (and  $\mathcal{K}$  is a witness to this). Otherwise, the prover selects an instance  $\psi$  of  $\text{Ax}(L)$  falsified in  $\mathcal{K}$  (there exists at least one);  $\psi$  is acknowledged as learned axiom and, after clausification, it is fed to the SAT-solver. We stress that a naive implementation of the procedure, where at each iteration of the main loop the computation restarts from scratch, would be highly inefficient: each time the SAT-solver should be initialized by inserting all the clauses encoding the input problem and all the clauses learned so far. Instead, we exploit an incremental SAT-solver, where clauses can be added but never deleted (hence, all the simplifications and optimisations performed by the solver are preserved); note that this prevents us from exploiting strategies based on standard sequent/tableaux calculi, where backtracking is required.

If the call `intuitRIL( $\alpha, L$ )` succeeds, by tracking the computation we get a derivation  $\mathcal{D}$  of  $\alpha$  in the sequent calculus  $C_L$  (see Fig. 1); from  $\mathcal{D}$  we can extract all the axioms learned during the computation. We stress that the procedure is quite modular: to handle a logic  $L$ , one has only to implement a specific learning mechanism for  $L$  (namely: if  $\mathcal{K}$  is not a model of  $L$ , pick an instance of  $\text{Ax}(L)$  falsified in  $\mathcal{K}$ ). The main drawback is that there is no general way to bound the learned axioms, thus termination must be investigated on a case-by-case basis. We guarantee termination for some relevant intermediate logics, such as Gödel-Dummett Logic GL, the family  $\text{GL}_n$  ( $n \geq 1$ ) of Gödel-Dummett Logics with depth bounded by  $n$  ( $\text{GL}_1$  coincides with Here and There Logic, well known for its applications in Answer Set Programming [15]) and Jankov Logic (for a presentation of such logics see [2]). As a corollary, for each of the mentioned logic  $L$  we get a bounding function [3], namely: given  $\alpha$ , we compute a bounded set  $\Psi_\alpha$  of instances of  $\text{Ax}(L)$  such that  $\alpha$  is valid in  $L$  iff  $\alpha$  is provable in IPL from assumptions  $\Psi_\alpha$ ; in general we improve the bounds in [1, 3]. The `intuitRIL` Haskell implementation and other additional material (e.g., the omitted proofs) can be downloaded at <https://github.com/cfiorentini/intuitRIL>.

## 2 Basic Definitions

Formulas, denoted by lowercase Greek letters, are built from an enumerable set of propositional variables  $\mathcal{V}$ , the constant  $\perp$  and the connectives  $\wedge, \vee, \rightarrow$ ; moreover,  $\neg\alpha$  stands for  $\alpha \rightarrow \perp$  and  $\alpha \leftrightarrow \beta$  stands for  $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ . Elements of the set  $\mathcal{V} \cup \{\perp\}$  are called *atoms* and are denoted by lowercase Roman letters,

uppercase Greek letters denote sets of formulas. By  $\mathcal{V}_\alpha$  we denote the set of propositional variables occurring in  $\alpha$ . The notation is extended to sets:  $\mathcal{V}_\Gamma$  is the union of  $\mathcal{V}_\alpha$  such that  $\alpha \in \Gamma$ ;  $\mathcal{V}_{\Gamma, \Gamma'}$  and  $\mathcal{V}_{\Gamma, \alpha}$  stand for  $\mathcal{V}_{\Gamma \cup \Gamma'}$  and  $\mathcal{V}_{\Gamma \cup \{\alpha\}}$  respectively. A *substitution* is a map from propositional variables to formulas. By  $[p_1 \mapsto \alpha_1, \dots, p_n \mapsto \alpha_n]$  we denote the substitution  $\chi$  such that  $\chi(p) = \alpha_i$  if  $p = p_i$  and  $\chi(p) = p$  otherwise; the set  $\{p_1, \dots, p_n\}$  is the *domain* of  $\chi$ , denoted by  $\text{Dom}(\chi)$ ;  $\epsilon$  is the substitution having empty domain. The application of  $\chi$  to a formula  $\alpha$ , denoted by  $\chi(\alpha)$ , is defined as usual;  $\chi(\Gamma)$  is the set of  $\chi(\alpha)$  such that  $\alpha \in \Gamma$ . The *composition*  $\chi_1 \cdot \chi_2$  is the substitution mapping  $p$  to  $\chi_1(\chi_2(p))$ .

A (*classical*) *interpretation*  $M$  is a subset of  $\mathcal{V}$ , identifying the propositional variables assigned to true. By  $M \models \alpha$  we mean that  $\alpha$  is true in  $M$ ;  $M \models \Gamma$  iff  $M \models \alpha$  for every  $\alpha \in \Gamma$ . Classical Propositional Logic (CPL) is the set of formulas true in every interpretation. We write  $\Gamma \vdash_c \alpha$  iff  $M \models \Gamma$  implies  $M \models \alpha$ , for every  $M$ . Note that  $\alpha$  is CPL-valid (namely,  $\alpha \in \text{CPL}$ ) iff  $\emptyset \vdash_c \alpha$ .

A (rooted) Kripke model is a quadruple  $\langle W, \leq, r, \vartheta \rangle$  where  $W$  is a finite and non-empty set (the set of *worlds*),  $\leq$  is a reflexive and transitive binary relation over  $W$ , the world  $r$  (the *root* of  $\mathcal{K}$ ) is the minimum of  $W$  w.r.t.  $\leq$ , and  $\vartheta : W \mapsto 2^\mathcal{V}$  (the *valuation* function) is a map obeying the persistence condition: for every pair of worlds  $w_1$  and  $w_2$  of  $\mathcal{K}$ ,  $w_1 \leq w_2$  implies  $\vartheta(w_1) \subseteq \vartheta(w_2)$ ; the triple  $\langle W, \leq, r \rangle$  is called (*Kripke*) *frame*. The valuation  $\vartheta$  is extended to a *forcing* relation between worlds and formulas as follows:

$$\begin{aligned} w \Vdash p &\text{ iff } p \in \vartheta(w), \forall p \in \mathcal{V} & w \nVdash \perp & \quad w \Vdash \alpha \wedge \beta \text{ iff } w \Vdash \alpha \text{ and } w \Vdash \beta \\ w \Vdash \alpha \vee \beta &\text{ iff } w \Vdash \alpha \text{ or } w \Vdash \beta & w \Vdash \alpha \rightarrow \beta &\text{ iff } \forall w' \geq w, w' \Vdash \alpha \text{ implies } w' \Vdash \beta. \end{aligned}$$

By  $w \Vdash \Gamma$  we mean that  $w \Vdash \alpha$  for every  $\alpha \in \Gamma$ . A formula  $\alpha$  is *valid* in the frame  $\langle W, \leq, r \rangle$  iff for every valuation  $\vartheta$ ,  $r \Vdash \alpha$  in the model  $\langle W, \leq, r, \vartheta \rangle$ . Propositional Intuitionistic Logic (IPL) is the set of formulas valid in all frames. Accordingly, if there is a model  $\mathcal{K}$  such that  $r \nVdash \alpha$  (here and below  $r$  designates the root of  $\mathcal{K}$ ), then  $\alpha$  is not IPL-valid; we call  $\mathcal{K}$  a *countermodel* for  $\alpha$ . We write  $\Gamma \vdash_i \delta$  iff, for every model  $\mathcal{K}$ ,  $r \Vdash \Gamma$  implies  $r \Vdash \delta$ ; thus,  $\alpha$  is IPL-valid iff  $\emptyset \vdash_i \alpha$ .

Let  $L$  be one of the logics IPL and CPL; then,  $L$  is closed under modus ponens ( $\{\alpha, \alpha \rightarrow \beta\} \subseteq L$  implies  $\beta \in L$ ) and under substitution (for every  $\chi$ ,  $\alpha \in L$  implies  $\chi(\alpha) \in L$ ). An *intermediate logic* is any set of formulas  $L$  such that  $\text{IPL} \subseteq L \subseteq \text{CPL}$ ,  $L$  is closed under modus ponens and under substitution. A model  $\mathcal{K}$  is an  $L$ -model iff  $r \Vdash L$ ; if  $r \nVdash \alpha$ , we say that  $\mathcal{K}$  is an  $L$ -*countermodel* for  $\alpha$ . An intermediate logic  $L$  can be characterized by a set of CPL-valid formulas, called the  $L$ -*axioms* and denoted by  $\text{Ax}(L)$ . An  $L$ -axiom  $\psi$  of  $\text{Ax}(L)$  must be understood as a schematic formula, representing all the formulas of the kind  $\chi(\psi)$ ; we call  $\chi(\psi)$  an *instance* of  $\psi$ . Formally,  $\text{IPL} + \text{Ax}(L)$  is the intermediate logic collecting the formulas  $\alpha$  such that  $\Psi \vdash_i \alpha$ , where  $\Psi$  is a finite set of instances of  $L$ -axioms from  $\text{Ax}(L)$ . A *bounding function* for  $L$  is a map that, given  $\alpha$ , yields a finite set  $\Psi_\alpha$  of instances of  $L$ -axioms such that  $\Psi_\alpha \vdash_i \alpha$ . If  $L$  admits a computable bounding function, we can reduce  $L$ -validity to IPL-validity (see [3] for an in-depth discussion). Let  $\mathcal{F}$  be a class of frames and let  $\text{Log}(\mathcal{F})$  be the set of formulas valid in all frames of  $\mathcal{F}$ ; then,  $\text{Log}(\mathcal{F})$  is an intermediate logic. A logic  $L$  has *Kripke semantics* iff there exists a class of frames  $\mathcal{F}$  such that  $L = \text{Log}(\mathcal{F})$ ; we also say that  $L$  is characterized by  $\mathcal{F}$ . Henceforth, when we

mention a logic  $L$ , we leave understood that  $L$  is an axiomatizable intermediate logic having Kripke semantics.

*Example 1 (GL).* A well-known intermediate logic is Gödel-Dummett logic GL [2], characterized by the class of linear frames. An axiomatization of GL is obtained by adding the linearity axiom **lin**  $= (a \rightarrow b) \vee (b \rightarrow a)$  to IPL. Using the terminology of [3], GL is formula-axiomatizable: a bounding function for GL is obtained by mapping  $\alpha$  to the set  $\Psi_\alpha$  of instances of **lin** where  $a$  and  $b$  are replaced with subformulas of  $\alpha$ . In [1] it is proved that it is sufficient to consider the subformulas of  $\alpha$  of the kind  $p \in \mathcal{V}_\alpha$ ,  $\neg\beta$ ,  $\beta_1 \rightarrow \beta_2$ . In Lemma 4 we further improve this bound tacking as bounding function the following map:

$$\begin{aligned} \text{Ax}_{\text{GL}}(\alpha) = & \{ (a \rightarrow b) \vee (b \rightarrow a) \mid a, b \in \mathcal{V}_\alpha \} \cup \{ (a \rightarrow \neg a) \vee (\neg a \rightarrow a) \mid a \in \mathcal{V}_\alpha \} \\ & \cup \{ (a \rightarrow (a \rightarrow b)) \vee ((a \rightarrow b) \rightarrow a) \mid a, b \in \mathcal{V}_\alpha \} \end{aligned}$$

Thus, if  $\mathcal{V}_\alpha = \{a\}$ , the only instance of **lin** to consider is  $(a \rightarrow \neg a) \vee (\neg a \rightarrow a)$ , independently of the size of  $\alpha$  (the other instances are IPL-valid and can be omitted). As pointed out in [3], GL is not variable-axiomatizable, namely: it is not sufficient to consider instances of **lin** obtained by replacing  $a$  and  $b$  with variables from  $\mathcal{V}_\alpha$ . As an example, let  $\alpha = \neg a \vee \neg\neg a$ ;  $\alpha$  is GL-valid, the only variable-replacement instance of **lin** is  $\psi_\alpha = (a \rightarrow a) \vee (a \rightarrow a)$  and  $\psi_\alpha \not\models_i \alpha$ .  $\diamond$

We review the main concepts about the clausification procedure described in [4]. *Clauses*  $\varphi$  and *implication clauses*  $\lambda$  are defined as

$$\begin{aligned} \varphi &:= \bigwedge A_1 \rightarrow \bigvee A_2 \mid \bigvee A_2 & \emptyset \subset A_k \subseteq \mathcal{V} \cup \{\perp\}, \text{fork} \in \{1, 2\} \\ \lambda &:= (a \rightarrow b) \rightarrow c & a \in \mathcal{V}, \{b, c\} \subseteq \mathcal{V} \cup \{\perp\} \end{aligned}$$

where  $\bigwedge A_1$  and  $\bigvee A_2$  denote the conjunction and the disjunction of the atoms in  $A_1$  and  $A_2$  respectively ( $\bigwedge \{a\} = \bigvee \{a\} = a$ ). Henceforth,  $\bigwedge \emptyset \rightarrow \bigvee A_2$  must be read as  $\bigvee A_2$ ;  $R, R_1, \dots$  denote sets of clauses,  $X, X_1, \dots$  sets of implication clauses. Given a set of implication clauses  $X$ , the *closure* of  $X$ , denoted by  $(X)^*$ , is the set of clauses  $b \rightarrow c$  such that  $(a \rightarrow b) \rightarrow c \in X$ .

The following lemma states some properties of clauses and closures.

**Lemma 1.** (i)  $R \vdash_i g$  iff  $R \vdash_c g$ , for every set of clauses  $R$  and every atom  $g$ .  
(ii)  $X \vdash_i b \rightarrow c$ , for every  $b \rightarrow c \in (X)^*$ .  
(iii)  $\Gamma \vdash_i \alpha$  iff  $\alpha \leftrightarrow g, \Gamma \vdash_i g$ , where  $g \notin \mathcal{V}_{\Gamma, \alpha}$ .

*Clausification.* We assume a procedure **Clausify** that, given a formula  $\alpha$ , computes sets of clauses  $R$  and  $X$  equivalent to  $\alpha$  w.r.t. IPL. Formally, let  $\alpha$  be a formula and let  $V$  be a set of propositional variables such that  $\mathcal{V}_\alpha \subseteq V$ . The procedure **Clausify**( $\alpha, V$ ) computes a triple  $(R, X, \chi)$  satisfying:

- (C1)  $\Gamma, \alpha \vdash_i \delta$  iff  $\Gamma, R, X \vdash_i \delta$ , for every  $\Gamma$  and  $\delta$  such that  $\mathcal{V}_{\Gamma, \delta} \subseteq V$ .
- (C2)  $\text{Dom}(\chi) = \mathcal{V}_{R, X} \setminus V$  and  $\mathcal{V}_{\chi(p)} \subseteq V$  for every  $p \in \text{Dom}(\chi)$ .
- (C3)  $R, X \vdash_i p \leftrightarrow \chi(p)$  for every  $p \in \text{Dom}(\chi)$ .

$$\begin{array}{c}
\frac{R \vdash_c g}{R, X \Rightarrow g} \text{cpl}_0 \quad \frac{R, A \vdash_c b \quad R, \varphi, X \Rightarrow g}{R, X \Rightarrow g} \text{cpl}_1(\lambda) \quad \begin{array}{l} \lambda = (a \rightarrow b) \rightarrow c \in X \\ A \subseteq \mathcal{V}_{R, X, g} \\ \varphi = \bigwedge (A \setminus \{a\}) \rightarrow c \end{array} \\
\\
\frac{R, (X)^*, X \Rightarrow g}{\Rightarrow \alpha} \text{Claus}_0(g, \chi) \quad \begin{array}{l} g \notin \mathcal{V}_\alpha \\ (R, X, \chi) = \text{Clausify}(\alpha \leftrightarrow g, \mathcal{V}_{\alpha, g}) \end{array} \\
\\
\frac{R, R', (X')^*, X, X' \Rightarrow g}{R, X \Rightarrow g} \text{Claus}_1(\psi, \chi) \quad \begin{array}{l} \psi \in \text{Ax}(L, \mathcal{V}_{R, X, g}) \\ (R', X', \chi) = \text{Clausify}(\psi, \mathcal{V}_{R, X, g}) \end{array} \\
\\
\begin{array}{l} R \text{ is a set of clauses} \\ X \text{ is a set of implication clauses} \\ g \text{ is an atom} \end{array} \quad \pi(\rho) = \begin{cases} \langle \emptyset, [g \mapsto \alpha] \cdot \chi \rangle & \text{if } \rho = \text{Claus}_0(g, \chi) \\ \langle \{\psi\}, \chi \rangle & \text{if } \rho = \text{Claus}_1(\psi, \chi) \\ \langle \emptyset, \epsilon \rangle & \text{otherwise} \end{cases}
\end{array}$$

**Fig. 1.** The sequent calculus  $C_L$ .

Basically, clausification introduces new propositional variables to represent subformulas of  $\alpha$ ; as a result we obtain a substitution  $\chi$  which tracks the mapping on the new variables. Condition (C1) states that  $\alpha$  can be replaced by  $R \cup X$  in IPL reasoning. By (C2) the domain of  $\chi$  consists of the new variables introduced in the clausification process. The following properties easily follow by (C1)–(C3):

$$(P1) \ R, X \vdash_i \alpha. \quad (P2) \ R, X \vdash_i \beta \leftrightarrow \chi(\beta) \text{ for every formula } \beta.$$

We exploit a **Clausify** procedure essentially similar to the one described in [4], with slight modifications in order to match (C3). As discussed in [4], in IPL we can use a weaker condition (either  $R, X \vdash_i p \rightarrow \chi(p)$  or  $R, X \vdash_i \chi(p) \rightarrow p$  according to the case). It is not obvious whether the weaker condition should be more efficient; in many cases strong equivalences are more performant, maybe because they trigger more simplifications in the SAT-solver.

*Example 2.* Let  $\alpha = (a \rightarrow b) \vee (b \rightarrow a)$  and  $V = \{a, b\}$ . The call **Clausify**( $\alpha, V$ ) introduces the new variables  $\tilde{p}_0$  and  $\tilde{p}_1$  associated with the subformulas  $a \rightarrow b$  and  $b \rightarrow a$  respectively. Accordingly, the obtained sets  $R$  and  $X$  must satisfy  $R, X \vdash_i \tilde{p}_0 \leftrightarrow (a \rightarrow b)$  and  $R, X \vdash_i \tilde{p}_1 \leftrightarrow (b \rightarrow a)$ . We get:

$$\begin{aligned}
R &= \{ \tilde{p}_0 \vee \tilde{p}_1, \tilde{p}_0 \wedge a \rightarrow b, \tilde{p}_1 \wedge b \rightarrow a \} & \chi &= [\tilde{p}_0 \mapsto a \rightarrow b, \tilde{p}_1 \mapsto b \rightarrow a] \\
X &= \{ (a \rightarrow b) \rightarrow \tilde{p}_0, (b \rightarrow a) \rightarrow \tilde{p}_1 \}
\end{aligned}$$

◇

### 3 The Calculus $C_L$

Let  $L$  be an intermediate logic; we introduce the sequent calculus  $C_L$  to prove  $L$ -validity. We assume that  $L$  is axiomatized by a set  $\text{Ax}(L)$  of  $L$ -axioms; by

$$\begin{array}{c}
\dfrac{\dots \quad \dfrac{R_{n-1} \vdash_c g}{R_{n-1}, X_{n-1} \Rightarrow g} \rho_n = \text{cpl}_0}{R_{n-2}, X_{n-2} \Rightarrow g} \rho_{n-1} \\
\vdots \\
\dfrac{\dots \quad \dfrac{R_1, X_1 \Rightarrow g}{R_0, X_0 \Rightarrow g} \rho_1}{\Rightarrow \alpha} \rho_0 = \text{Claus}_0
\end{array}
\quad
\begin{array}{l}
\forall i \in \{1, \dots, n-1\}, \rho_i = \text{cpl}_1 \text{ or } \rho_i = \text{Claus}_1 \\
\pi(\mathcal{D}) = \langle \Psi_0 \cup \dots \cup \Psi_n, \chi_0 \cdot \dots \cdot \chi_n \rangle \\
\text{where } \langle \Psi_j, \chi_j \rangle = \pi(\rho_j)
\end{array}$$

**Fig. 2.** A  $C_L$ -derivation of  $\Rightarrow \alpha$ .

$\text{Ax}(L, V)$  we denote the set of instances  $\psi$  of  $L$ -axioms such that  $\mathcal{V}_\psi \subseteq V$ . The calculus relies on a clausification procedure **Clausify** satisfying conditions (C1)–(C3) and acts on sequents  $\Gamma \Rightarrow \delta$  such that:

- either  $\Gamma = \emptyset$  or  $\Gamma = R \cup X$  and  $(X)^* \subseteq R$  and  $\delta$  is an atom.

Rules of  $C_L$  are displayed in Fig. 1. Rule  $\text{cpl}_0$  (initial rule) can only be applied if the condition  $R \vdash_c g$  holds; if this is the case, the conclusion  $R, X \Rightarrow g$  is an initial sequent, namely a top sequent of a derivation. The other rules depend on parameters that are made explicit in the rule name. A bottom-up application of  $\text{cpl}_1$  requires the choice of an implication clause  $\lambda = (a \rightarrow b) \rightarrow c$  from  $X$ , we call the *main formula*, and the selection of a set of atoms  $A \subseteq \mathcal{V}_{R, X, g}$  such that  $R, A \vdash_c b$ , where  $b$  is the middle variable in  $\lambda$ . As discussed in [8, 9],  $\text{cpl}_1$  is a sort of generalization of the rule  $L \rightarrow \rightarrow$  of the sequent calculus LJ/T/G4ip for IPL [5, 18]. Rules  $\text{Claus}_0$  and  $\text{Claus}_1$  exploit the clausification procedure. Rule  $\text{Claus}_0$  requires the clausification of the formula  $\alpha \leftrightarrow g$ , with  $g$  a new atom ( $g \notin \mathcal{V}_\alpha$ ); in rule  $\text{Claus}_1$ , the clausified formula  $\psi$  is selected from  $\text{Ax}(L, \mathcal{V}_{R, X, g})$ . In both cases, the clauses returned by **Clausify** are stored in the premise of the applied rule and the computed substitution  $\chi$  is displayed in the rule name; moreover,  $\text{Claus}_0$  is annotated with the new atom  $g$  and  $\text{Claus}_1$  with the chosen  $L$ -axiom  $\psi$ . To recover the relevant information associated with the application of a rule  $\rho$ , in Fig. 1 we define the pair  $\pi(\rho) = \langle \Psi, \chi \rangle$ , where  $\Psi$  is a set of instances of  $L$ -axioms and  $\chi$  is a substitution.  $C_L$ -trees and  $C_L$ -derivations are defined as usual (see e.g. [18]); a sequent  $\sigma$  is provable in  $C_L$  iff there exists a  $C_L$ -derivation having root sequent  $\sigma$ . Let us consider a  $C_L$ -derivation  $\mathcal{D}$  of  $\Rightarrow \alpha$  (see Fig. 2). Reading the derivation bottom-up, the first applied rule is  $\text{Claus}_0$ . After such an application, the obtained sequents have the form  $\sigma_k = R_k, X_k \Rightarrow g$ , where  $R_k \cup X_k$  is non-empty, thus rule  $\text{Claus}_0$  cannot be applied any more; the rule applied at the top is  $\text{cpl}_0$ . Note that  $\mathcal{D}$  contains a unique branch, consisting of the sequents  $\Rightarrow \alpha, \sigma_0, \dots, \sigma_{n-1}$ . In Fig. 2 we also define the pair  $\pi(\mathcal{D}) = \langle \Psi, \chi \rangle$ :  $\Psi$  collects the (instances of)  $L$ -axioms selected by rule  $\text{Claus}_1$ ,  $\chi$  is obtained by composing the substitutions associated with the applied rules. The definition of  $\pi(\mathcal{T})$ , with  $\mathcal{T}$  a  $C_L$ -tree, is similar. By  $\mathcal{T}(\alpha; R, X \Rightarrow g)$  we denote a  $C_L$ -tree having root  $\Rightarrow \alpha$  and leaf  $R, X \Rightarrow g$ . Given a  $C_L$ -tree  $\mathcal{T}$ ,  $\mathcal{V}_\mathcal{T}$  is the set of variables occurring in  $\mathcal{T}$ . We state some properties about  $C_L$ -trees:

**Lemma 2.** Let  $\mathcal{T} = \mathcal{T}(\alpha; R, X \Rightarrow g)$  and let  $\pi(\mathcal{T}) = \langle \Psi, \chi \rangle$ .

- (i)  $\mathcal{V}_{\chi(p)} \subseteq \mathcal{V}_\alpha$ , for every  $p \in \mathcal{V}_\mathcal{T}$ .
- (ii)  $R, X \vdash_i \beta \leftrightarrow \chi(\beta)$ , for every formula  $\beta$ .
- (iii) If  $R, X, \Gamma \vdash_i g$  and  $\mathcal{V}_\Gamma \subseteq \mathcal{V}_\alpha$ , then  $\Gamma, \chi(\Psi) \vdash_i \alpha$ .

**Proposition 1.** Let  $\mathcal{D}$  be a  $C_L$ -derivation of  $\Rightarrow \alpha$  and let  $\pi(\mathcal{D}) = \langle \Psi, \chi \rangle$ . Then,  $\mathcal{V}_{\chi(\Psi)} \subseteq \mathcal{V}_\alpha$  and  $\chi(\Psi) \vdash_i \alpha$ .

*Proof.* Since  $\mathcal{D}$  is a  $C_L$ -derivation,  $\mathcal{D}$  has the form depicted on the right where  $\mathcal{T} = \mathcal{T}(\alpha; R, X \Rightarrow g)$ ; note that  $\pi(\mathcal{T}) = \pi(\mathcal{D}) = \langle \Psi, \chi \rangle$ . Since  $R \vdash_c g$ , by Lemma 1(i) we get  $R \vdash_i g$ , hence  $R, X \vdash_i g$ . We can apply Lemma 2 and claim that  $\mathcal{V}_{\chi(\Psi)} \subseteq \mathcal{V}_\alpha$  and  $\chi(\Psi) \vdash_i \alpha$ .  $\square$

$$\mathcal{D} = \frac{R \vdash_c g}{R, X \Rightarrow g} \text{cpl}_0 \quad \vdots \quad \mathcal{T} \Rightarrow \alpha$$

Given a  $C_L$ -derivation  $\mathcal{D}$  of  $\Rightarrow \alpha$ , Prop. 1 exhibits how to extract a set of instances  $\Psi_\alpha$  of the  $L$ -axioms such that  $\Psi_\alpha \vdash_i \alpha$ . If  $\mathcal{D}$  does not contain applications of rule Claus<sub>1</sub>,  $\Psi_\alpha$  is empty, and this ascertains that  $\alpha$  is IPL-valid; actually,  $\mathcal{D}$  can be immediately embedded into the calculus for IPL introduced in [8]. As an immediate consequence of Prop. 1, we get the soundness of  $C_L$ : if  $\Rightarrow \alpha$  is provable in  $C_L$ , then  $\alpha$  is  $L$ -valid.

Even though  $C_L$ -derivations have a simple structure, the design of a root-first proof search strategy for  $C_L$  is far from being trivial. After having applied rule Claus<sub>0</sub> to the root sequent  $\Rightarrow \alpha$ , we enter a loop where at each iteration  $k$  we search for a derivation of  $\sigma_k = R_k, X_k \Rightarrow g$ . It is convenient to firstly check whether  $R_k \vdash_c g$  so that, by applying rule cpl<sub>0</sub>, we immediately close the derivation at hand. To check classical provability, we exploit a SAT-solver; each time the solver is invoked, the set  $R_k$  has increased, thus it is advantageous to use an incremental SAT-solver. If  $R_k \not\vdash_c g$ , we have to apply either rule cpl<sub>1</sub> or rule Claus<sub>1</sub>, but it is not obvious which strategy should be followed. First, we have to select one between the two rules. If rule cpl<sub>1</sub> is chosen, we have to guess proper  $\lambda$  and  $A$ ; otherwise, we have to apply Claus<sub>1</sub>, and this requires the selection of an instance  $\psi$  of an  $L$ -axiom. In any case, if we followed a blind choice, the procedure would be highly inefficient. To guide proof search, we follow a different approach based on countermodel construction; to this aim, we introduce a representation of Kripke models where worlds are classical interpretations ordered by inclusion.

*Countermodels.* Let  $W$  be a finite set of interpretations with minimum  $M_0$ , namely:  $M_0 \subseteq M$  for every  $M \in W$ . By  $\mathcal{K}(W)$  we denote the Kripke model  $\langle W, \leq, M_0, \vartheta \rangle$  where  $\leq$  coincides with the subset relation  $\subseteq$  and  $\vartheta$  is the identity map, thus  $M \Vdash p$  (in  $\mathcal{K}(W)$ ) iff  $p \in M$ . We introduce the following *realizability relation*  $\triangleright_W$  between elements of  $W$  and implication clauses:

$$M \triangleright_W (a \rightarrow b) \rightarrow c \text{ iff } (a \in M) \text{ or } (b \in M) \text{ or } (c \in M) \text{ or } \\ (\exists M' \in W \text{ s.t. } M \subset M' \text{ and } a \in M' \text{ and } b \notin M').$$

By  $M \triangleright_W X$  we mean that  $M \triangleright_W \lambda$  for every  $\lambda \in X$ . We state the crucial properties of the model  $\mathcal{K}(W)$ :

**Proposition 2.** *Let  $\mathcal{K}(W)$  be the model generated by  $W$  and let  $w \in W$ . Let  $\varphi$  be a clause and  $\lambda = (a \rightarrow b) \rightarrow c$  an implication clause.*

- (i) *If  $w' \models \varphi$ , for every  $w' \in W$  such that  $w \leq w'$ , then  $w \models \varphi$ .*
- (ii) *If  $w' \models b \rightarrow c$  and  $w' \triangleright_W \lambda$ , for every  $w' \in W$  such that  $w \leq w'$ , then  $w \models \lambda$ .*

Let  $\mathcal{K}(W)$  be a model with root  $r$ , and assume that every interpretation  $w$  in  $W$  is a model of  $R$ ; our goal is to get  $r \models R \cup X$  (where  $(X)^* \subseteq R$ ), possibly by filling  $W$  with new worlds. To this aim, we exploit Prop. 2. By our assumption and point (i), we claim that  $r \models R$ . Suppose that there is  $w \in W$  and  $\lambda = (a \rightarrow b) \rightarrow c \in X$  such that  $w \not\models_W \lambda$ ; is it possible to amend  $\mathcal{K}(W)$  in order to match (ii) and conclude  $r \models X$ ? By definition of  $\triangleright_W$ , none of the atoms  $a, b, c$  belongs to  $w$ ; moreover  $\mathcal{K}(W)$  lacks a world  $w'$  such that  $w \subset w'$  and  $a \in w'$  and  $b \notin w'$ . We can try to fix  $\mathcal{K}(W)$  by inserting the missing world  $w'$ ; to preserve (i), we also need  $w' \models R$ . Accordingly, such a  $w'$  exists if and only if  $R, w, a \not\models_c b$ . This can be checked by querying a SAT-solver; moreover, if  $R, w, a \not\models_c b$ , the solver also computes the required  $w'$ . This completion process must be iterated until  $\mathcal{K}(W)$  has been saturated with all the missing worlds or we get stuck. It is easy to check that the process eventually terminates. This is one of the key ideas beyond the procedure `intuitRIL` we present in next section.

## 4 The Procedure `intuitRIL`

We present the procedure `intuitRIL` (`intuit` with Restart for Intermediate Logics) that, given a formula  $\alpha$  and a logic  $L = \text{IPL} + \text{Ax}(L)$ , returns either a set of  $L$ -axioms  $\Psi_\alpha$  or a model  $\mathcal{K}(W)$  with the following properties:

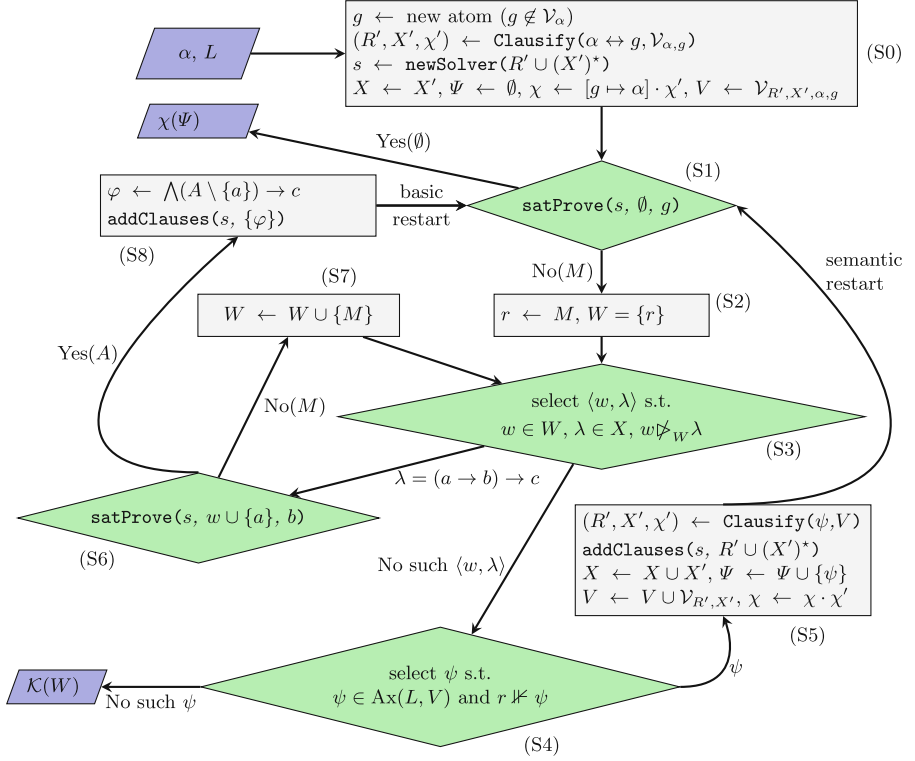
- (Q1) If `intuitRIL`( $\alpha, L$ ) returns  $\Psi_\alpha$ , then  $\Psi_\alpha \subseteq \text{Ax}(L, \mathcal{V}_\alpha)$  and  $\Psi_\alpha \vdash_i \alpha$ .
- (Q2) If `intuitRIL`( $\alpha, L$ ) returns  $\mathcal{K}(W)$ , then  $\mathcal{K}(W)$  is an  $L$ -countermodel for  $\alpha$ .

Thus,  $\alpha$  is  $L$ -valid in the former case, not  $L$ -valid in the latter. If `intuitRIL`( $\alpha, L$ ) returns  $\Psi_\alpha$ , by tracing the computation we can build a  $C_L$ -derivation  $\mathcal{D}$  of  $\Rightarrow \alpha$  such that  $\Psi_\alpha = \chi(\Psi)$ , where  $\langle \Psi, \chi \rangle = \pi(\mathcal{D})$ ; this certifies that  $\Psi_\alpha \vdash_i \alpha$ .

The procedure is described by the flowchart in Fig. 3 and exploits a single incremental SAT-solver  $s$ : clauses can be added to  $s$  but not removed; by  $R(s)$  we denote the set of clauses stored in  $s$ . The SAT-solver is required to support the following operations:

- `newSolver`( $R$ ) creates a new SAT-solver initialized with the clauses in  $R$ .
- `addClauses`( $s, R$ ) adds the clauses in  $R$  to the SAT-solver  $s$ .
- `satProve`( $s, A, g$ ) calls  $s$  to decide whether  $R(s), A \vdash_c g$  ( $A$  is a set of propositional variables). The solver outputs one of the following answers:
  - Yes( $A'$ ): thus,  $A' \subseteq A$  and  $R(s), A' \vdash_c g$ ;
  - No( $M$ ): thus,  $A \subseteq M \subseteq \mathcal{V}_{R(s)} \cup A$  and  $M \models R(s)$  and  $g \notin M$ .

In the former case it follows that  $R(s), A \vdash_c g$ , in the latter  $R(s), A \not\models_c g$ .



**Fig. 3.** Computation of  $\text{intuitRIL}(\alpha, L)$ .

The computation of  $\text{intuitRIL}(\alpha, L)$  consists of the following steps:

- (S0) The formula  $\alpha \leftrightarrow g$ , with  $g$  new propositional variable, is clausified. The outcome  $(R', X', \chi')$  is used to create a new SAT-solver  $s$  and to properly initialize the global variables  $X$  (set of implication clauses),  $\Psi$  (set of  $L$ -axiom instances),  $V$  (set of propositional variables) and  $\chi$  (substitution).
- (S1) A loop starts (*main loop*). The SAT-solver  $s$  is called to check whether  $R(s) \vdash_c g$ . If the answer is  $\text{Yes}(\emptyset)$ , the computation stops yielding  $\chi(\Psi)$ . Otherwise, the output is  $\text{No}(M)$  and the computation continues at Step (S2).
- (S2) We set  $r = M$  (the root of  $\mathcal{K}(W)$ ) and  $W = \{r\}$ .
- (S3) A loop starts (*inner loop*). We have to select a pair  $\langle w, \lambda \rangle$  such that  $w \in W$ ,  $\lambda \in X$  and  $w \not\models_W \lambda$ . If such a pair does not exist, the inner loop ends and next step is (S4), otherwise the inner loop continues at Step (S6).
- (S4) As we show in Lemma 3, at this point  $\mathcal{K}(W)$  is a countermodel for  $\alpha$ . If all the axioms in  $\text{Ax}(L, V)$  are forced at the root  $r$  of  $\mathcal{K}(W)$ , then  $\mathcal{K}(W)$  is an  $L$ -countermodel for  $\alpha$  and the computation ends returning  $\mathcal{K}(W)$ . Otherwise, we select  $\psi$  from  $\text{Ax}(L, V)$  such that  $r \not\models \psi$  and the computation continues at Step (S5); we call  $\psi$  the *learned axiom*.



- (S5) We clausify  $\psi$  and we update the global variables. The computation restarts from Step (S1) with a new iteration of the main loop (*semantic restart*).
- (S6) Let  $\langle w, (a \rightarrow b) \rightarrow c \rangle$  be the pair selected at Step (S3). The SAT-solver  $s$  is called to check whether  $R(s), w, a \vdash_c b$ . If the result is  $\text{No}(M)$ , the inner loop continues at step (S7). Otherwise, the answer is  $\text{Yes}(A)$ ; the inner loop ends and the computation continues at Step (S8).
- (S7) The interpretation  $M$  is added to  $W$  and the computation continues at Step (S3) with a new iteration of the inner loop.
- (S8) The clause  $\varphi$  (*learned basic clause*) is added to the SAT-solver  $s$  and the computation restarts from Step (S1) (*basic restart*).

Intuitively,  $\text{intuitRIL}(\alpha, L)$  searches for an  $L$ -countermodel  $\mathcal{K}(W)$  for  $\alpha$ . In the construction of  $\mathcal{K}(W)$ , whenever a conflict arises, a restart operation is triggered. A basic restart happens when it is not possible to fill the set  $W$  with a missing world (see the discussion after Prop. 2). A semantic restart is thrown when  $\mathcal{K}(W)$  is a countermodel for  $\alpha$  but it fails to be an  $L$ -model. In either case, the construction of  $\mathcal{K}(W)$  restarts from scratch. However, to prevent that the same kind of conflict shows up again, new clauses are learned and fed to the SAT-solver (this complies with  $\text{DPLL}(\mathcal{T})$  with learning computation paradigm [16]). If the outcome is  $\chi(\Psi)$ , by tracing the computation we can build a  $C_L$ -derivation  $\mathcal{D}$  of  $\Rightarrow \alpha$  such that  $\pi(\mathcal{D}) = \langle \Psi, \chi \rangle$ . The derivation is built bottom-up. The initial Step (S0) corresponds to the application of rule  $\text{Claus}_0$  to the root sequent  $\Rightarrow \alpha$ ; basic and semantic restarts bottom-up expand the derivation by applying rule  $\text{cpl}_1$  and  $\text{Claus}_1$  respectively. We stress that the procedure is quite modular; to treat a specific logic  $L$  one has only to provide a concrete implementation of Step (S4). For  $L = \text{IPL}$ , Step (S4) is trivial, since the set  $\text{Ax}(\text{IPL}, V)$  is empty. Actually,  $\text{intuitRIL}$  applied to IPL has the same behaviour as the procedure  $\text{intuitR}$  introduced in [8].

*Example 3.* Let us consider *Jankov axiom*  $\mathbf{wem} = \neg a \vee \neg \neg a$  [2, 13] (aka *weak excluded middle*), which holds in all frames having a single maximal world (thus,  $\mathbf{wem}$  is GL-valid). The trace of the execution of  $\text{intuitRIL}(\mathbf{wem}, \text{GL})$  is shown in Fig. 4. The initial clausification yields  $(R_0, X_0, \tilde{g})$ , where  $X_0$  consists of the implication clauses  $\lambda_0, \lambda_1$  in Fig. 4 and  $R_0$  contains the 7 clauses below:

$$\tilde{g} \rightarrow \tilde{p}_2, \quad \tilde{p}_0 \rightarrow \tilde{p}_2, \quad a \wedge \tilde{p}_0 \rightarrow \perp, \quad \tilde{p}_1 \rightarrow \tilde{p}_2, \quad \tilde{p}_0 \wedge \tilde{p}_1 \rightarrow \perp, \quad \tilde{p}_2 \rightarrow \tilde{g}, \quad \tilde{p}_2 \rightarrow \tilde{p}_0 \vee \tilde{p}_1.$$

Each row in Fig. 4 displays the validity tests performed by the SAT-solver and the computed answers. If the result is  $\text{No}(M)$ , the last two columns show the worlds  $w_k$  in the current set  $W$  and, for each  $w_k$ , the list of  $\lambda$  such that  $w_k \not\vdash_W \lambda$ ; the pair selected for the next step is underlined. For instance, after call (1) we have  $W = \{w_0\}$ ,  $w_0 \not\vdash_W \lambda_0$  and  $w_0 \not\vdash_W \lambda_1$ ; the selected pair is  $\langle w_0, \lambda_0 \rangle$ . After call (2), the set  $W$  is updated by adding the world  $w_1$ ; we have  $w_1 \supset_W \lambda_0$ ,  $w_1 \supset_W \lambda_1$ ,  $w_0 \supset_W \lambda_0$  and  $w_0 \not\vdash_W \lambda_1$ . Whenever the SAT-solver outputs  $\text{Yes}(A)$ , we display the learned clause  $\psi_k$ . The SAT-solver is invoked 18 times and there are 6 restarts (1 semantic, 5 basic). After (3), we get  $W = \{w_0, w_1, w_2\}$  and no pair  $\langle w, \lambda \rangle$  can be selected, hence the model  $\mathcal{K}(W)$  (displayed in the figure) is

a countermodel for **wem**. However,  $\mathcal{K}(W)$  is not a GL-model (indeed, it is not linear), hence we choose an instance of the linearity axiom not forced at  $w_0$ , namely  $\psi_0$ , and we force a semantic restart. The clausification of  $\psi_0$  produces 6 new clauses and the new implication clauses  $\lambda_2, \lambda_3, \lambda_4$ . After each restart, the sets  $R_j$  are:

$$\begin{aligned} R_1 &= R_0 \cup \{ \tilde{p}_3 \rightarrow \tilde{p}_4, a \rightarrow \tilde{p}_5, \tilde{p}_3 \wedge \tilde{p}_5 \rightarrow a, a \wedge \tilde{p}_4 \rightarrow \tilde{p}_3, a \wedge \tilde{p}_3 \rightarrow \perp, \tilde{p}_4 \vee \tilde{p}_5 \} \\ R_j &= R_{j-1} \cup \{ \psi_{j-1} \} \quad \text{for } 2 \leq j \leq 6 \text{ (the } \psi'_j \text{'s are defined in Fig. 4).} \end{aligned}$$

The  $C_{\text{GL}}$ -derivation of  $\Rightarrow \neg a \vee \neg \neg a$  extracted from the computation is:

$$\begin{array}{c} \frac{R_1, a, \tilde{p}_0 \vdash_c \perp}{R_1, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_3) \\ \frac{\frac{R_2, a, \tilde{p}_0 \vdash_c \perp}{R_2, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_0) \quad \frac{R_3, a, \tilde{p}_3 \vdash_c \perp}{R_3, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_1)}{\frac{R_4, \tilde{p}_0, \tilde{p}_5 \vdash_c \perp}{R_4, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_1)} \text{cpl}_1(\lambda_1) \\ \frac{\frac{R_5, a, \tilde{p}_4 \vdash_c \perp}{R_5, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_0) \quad \frac{R_6, X_1 \Rightarrow \tilde{g}}{R_6 \vdash_c \tilde{g}} \text{cpl}_0}{\frac{R_5, a, \tilde{p}_4 \vdash_c \perp}{R_5, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_1)} \text{cpl}_1(\lambda_1) \\ \frac{\frac{R_3, a, \tilde{p}_3 \vdash_c \perp}{R_3, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_1) \quad \frac{R_4, \tilde{p}_0, \tilde{p}_5 \vdash_c \perp}{R_4, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_1)}{\frac{R_2, a, \tilde{p}_0 \vdash_c \perp}{R_2, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_0)} \text{cpl}_1(\lambda_1) \\ \frac{\frac{R_1, a, \tilde{p}_0 \vdash_c \perp}{R_1, X_1 \Rightarrow \tilde{g}} \text{cpl}_1(\lambda_3) \quad \frac{R_2, X_1 \Rightarrow \tilde{g}}{R_2 \vdash_c \tilde{g}} \text{cpl}_1(\lambda_0)}{\frac{R_1, X_1 \Rightarrow \tilde{g}}{R_0, X_0 \Rightarrow \tilde{g}} \text{Claus}_1(\psi_0, \chi_1)} \text{Claus}_0(\tilde{g}, \chi_0) \\ \frac{R_0, X_0 \Rightarrow \tilde{g}}{\Rightarrow \neg a \vee \neg \neg a} \end{array}$$

◇

Now, we discuss partial correctness and termination of **intuitRIL**. Let us denote with  $\sim_c$  classical equivalence ( $\alpha \sim_c \beta$  iff  $\vdash_c \alpha \leftrightarrow \beta$ ) and with  $\sim_i$  intuitionistic equivalence ( $\alpha \sim_i \beta$  iff  $\vdash_i \alpha \leftrightarrow \beta$ ). We introduce some notation.

- (†) The following terms refer to the configuration at the beginning of iteration  $k$  ( $k \geq 0$ ), just after the execution of Step (S2):
- $\Phi_k$  is the set collecting all the learned basic clauses;
  - $R_k$  is the set of clauses stored in the SAT-solver  $s$ ;
  - $X_k, \Psi_k, V_k, \chi_k, r_k$  are the values of the corresponding global variables.

In Fig. 5 we inductively define the  $C_L$ -tree  $\mathcal{T}_k$ , having the form  $\mathcal{T}(\alpha; R_k, X_k \Rightarrow g)$ . In the application of rule  $\text{Claus}_0$ ,  $g$  and  $\chi'$  are defined as in Step (S0). In rule  $\text{cpl}_1$ ,  $\lambda$  is the implication clause selected at iteration  $k-1$  (of the main loop) in the last execution of Step (S3);  $A$  is the value computed at Step (S6) of iteration  $k-1$ . In the application of rule  $\text{Claus}_1$ ,  $\psi$  and  $\chi'$  are defined as in the execution of Step (S4) and (S5) of iteration  $k-1$ . One can easily check that the applications of the rules are sound. If Step (S1) yields  $\text{Yes}(\emptyset)$ , we can turn  $\mathcal{T}_k$  into a  $C_L$ -derivation by applying rule  $\text{cpl}_0$ .

Next lemma states some relevant properties of the computations of **intuitRIL**.

**Lemma 3.** *Let us consider the execution of iteration  $k$  of the main loop ( $k \geq 0$ ).*

- (i)  $(X_k)^* \cup \Phi_k \subseteq R_k$ .
- (ii)  $V_k = \mathcal{V}_{\mathcal{T}_k}$  and  $\Psi_k \subseteq \text{Ax}(L, V_k)$  and  $\pi(\mathcal{T}_k) = \langle \Psi_k, \chi_k \rangle$ .
- (iii)  $\mathcal{V}_{\chi_k(p)} \subseteq \mathcal{V}_\alpha$ , for every  $p \in V_k$ , and  $R_k, X_k \vdash_i \beta \leftrightarrow \chi_k(\beta)$ , for every  $\beta$ .

$$\begin{aligned}
\lambda_0 &= (\tilde{p}_0 \rightarrow \perp) \rightarrow \tilde{p}_1 & \lambda_1 &= (a \rightarrow \perp) \rightarrow \tilde{p}_0 \\
\lambda_2 &= (a \rightarrow \tilde{p}_3) \rightarrow \tilde{p}_4 & \lambda_3 &= (a \rightarrow \perp) \rightarrow \tilde{p}_3 & \lambda_4 &= (\tilde{p}_3 \rightarrow a) \rightarrow \tilde{p}_5 \\
w_0 &= \emptyset & w_1 &= \{\tilde{g}, \tilde{p}_0, \tilde{p}_2\} & w_2 &= \{a, \tilde{g}, \tilde{p}_1, \tilde{p}_2\} & w_3 &= \{\tilde{p}_4\} & w_4 &= \{\tilde{g}, \tilde{p}_0, \tilde{p}_2, \tilde{p}_4\} \\
w_5 &= \{\tilde{g}, \tilde{p}_0, \tilde{p}_2, \tilde{p}_3, \tilde{p}_4\} & w_6 &= \{a, \tilde{p}_5\} & w_7 &= \{\tilde{p}_3, \tilde{p}_4\} & w_8 &= \{\tilde{g}, \tilde{p}_0, \tilde{p}_2, \tilde{p}_3, \tilde{p}_4\} \\
w_9 &= \{\tilde{p}_5\} & w_{10} &= \{\tilde{p}_4\} & w_{11} &= \{\tilde{g}, \tilde{p}_0, \tilde{p}_2, \tilde{p}_3, \tilde{p}_4\} \\
\chi_0 &= [\tilde{g} \mapsto \neg a \vee \neg \neg a, \tilde{p}_0 \mapsto \neg a, \tilde{p}_1 \mapsto \neg \neg a, \tilde{p}_2 \mapsto \neg a \vee \neg \neg a] \\
\chi_1 &= [\tilde{p}_3 \mapsto \neg a, \tilde{p}_4 \mapsto a \rightarrow \neg a, \tilde{p}_5 \mapsto \neg a \rightarrow a]
\end{aligned}$$

	@SAT	Answer	$W$	$\lambda$ s.t. $w \not\vdash_W \lambda$
<b>Start</b>	(1) $R_0 \vdash_c \tilde{g} ?$	No( $w_0$ )	$w_0$	$\lambda_0, \lambda_1$
	(2) $R_0, w_0, \tilde{p}_0 \vdash_c \perp ?$	No( $w_1$ )	$w_1$ $w_0$	$\emptyset$ $\lambda_1$
	(3) $R_0, w_0, a \vdash_c \perp ?$	No( $w_2$ )	$w_2$ $w_1$ $w_0$	$\emptyset$ $\emptyset$ $\emptyset$

Semantic  
failure

$w_1 : \tilde{g}, \tilde{p}_0, \tilde{p}_2$   
 $w_0 : \emptyset$

$w_2 : a, \tilde{g}, \tilde{p}_1, \tilde{p}_2$   
 $w_0 : \emptyset$

**Learned axiom:**

$$\psi_0 = (a \rightarrow \neg a) \vee (\neg a \rightarrow a)$$

<b>SRest 1</b>	(4) $R_1 \vdash_c \tilde{g} ?$	No( $w_3$ )	$w_3$	$\lambda_0, \lambda_1, \lambda_3, \lambda_4$
	(5) $R_1, w_3, \tilde{p}_0 \vdash_c \perp ?$	No( $w_4$ )	$w_4$ $w_3$	$\lambda_3, \lambda_4$ $\lambda_1, \lambda_3, \lambda_4$
	(6) $R_1, w_4, \tilde{p}_3 \vdash_c a ?$	No( $w_5$ )	$w_5$ $w_4$ $w_3$	$\emptyset$ $\lambda_3$ $\lambda_1, \lambda_3$
	(7) $R_1, w_4, a \vdash_c \perp ?$	Yes( $\{a, \tilde{p}_0\}$ )	$\psi_1 = \tilde{p}_0 \rightarrow \tilde{p}_3$	
<b>BRest 2</b>	(8) $R_2 \vdash_c \tilde{g} ?$	No( $w_6$ )	$w_6$	$\lambda_0$
	(9) $R_2, w_6, \tilde{p}_0 \vdash_c \perp ?$	Yes( $\{a, \tilde{p}_0\}$ )	$\psi_2 = a \rightarrow \tilde{p}_1$	
<b>BRest 3</b>	(10) $R_3 \vdash_c \tilde{g} ?$	No( $w_7$ )	$w_7$	$\lambda_0, \lambda_1$
	(11) $R_3, w_7, \tilde{p}_0 \vdash_c \perp ?$	No( $w_8$ )	$w_8$ $w_7$	$\emptyset$ $\lambda_1$
	(12) $R_3, w_7, a \vdash_c \perp ?$	Yes( $\{a, \tilde{p}_3\}$ )	$\psi_3 = \tilde{p}_3 \rightarrow \tilde{p}_0$	
<b>BRest 4</b>	(13) $R_4 \vdash_c \tilde{g} ?$	No( $w_9$ )	$w_9$	$\lambda_0, \lambda_1, \lambda_2, \lambda_3$
	(14) $R_4, w_9, \tilde{p}_0 \vdash_c \perp ?$	Yes( $\{\tilde{p}_0, \tilde{p}_5\}$ )	$\psi_4 = \tilde{p}_5 \rightarrow \tilde{p}_1$	
<b>BRest 5</b>	(15) $R_5 \vdash_c \tilde{g} ?$	No( $w_{10}$ )	$w_{10}$	$\lambda_0, \lambda_1, \lambda_3, \lambda_4$
	(16) $R_5, w_{10}, \tilde{p}_0 \vdash_c \perp ?$	No( $w_{11}$ )	$w_{11}$ $w_{10}$	$\emptyset$ $\lambda_1, \lambda_3$
	(17) $R_5, w_{10}, a \vdash_c \perp ?$	Yes( $\{a, \tilde{p}_4\}$ )	$\psi_5 = \tilde{p}_4 \rightarrow \tilde{p}_0$	
<b>BRest 6</b>	(18) $R_6 \vdash_c \tilde{g} ?$	Yes( $\emptyset$ )	<b>Proved</b>	

**Fig. 4.** Computation of  $\text{intuitRIL}(\neg a \vee \neg \neg a, \text{GL})$ .

$$\begin{aligned}
\mathcal{T}_0 &= \frac{R_0, X_0 \Rightarrow g}{\Rightarrow \alpha} \text{Claus}_0(g, \chi') \\
\mathcal{T}_k &= \frac{\frac{R_{k-1}, A \vdash_c b \quad R_k, X_k \Rightarrow g}{R_{k-1}, X_{k-1} \Rightarrow g} \text{cpl}_1(\lambda)}{\vdots \quad \mathcal{T}_{k-1} \Rightarrow \alpha} \quad \frac{R_k, X_k \Rightarrow g}{R_{k-1}, X_{k-1} \Rightarrow g} \text{Claus}_1(\psi, \chi') \\
&\quad \vdots \quad \mathcal{T}_{k-1} \Rightarrow \alpha
\end{aligned}$$

if  $k > 0$  and iteration  $k - 1$  ends with a basic restart (thus  $X_k = X_{k-1}$ )

if  $k > 0$  and iteration  $k - 1$  ends with a semantic restart

**Fig. 5.** Definition of  $\mathcal{T}_k$  ( $k \geq 0$ ).

- (iv) At every step after (S2),  $w \models R_k$ , for every  $w \in W$ .
- (v) At every step after (S2),  $r_k$  is the root of  $\mathcal{K}(W)$  and  $r_k \Vdash R_k$  and  $r_k \not\models g$ .
- (vi) At Step (S4),  $r_k \Vdash R_k \cup X_k \cup \Psi_k$  and  $r_k \not\models g$  (in  $\mathcal{K}(W)$ ).
- (vii) Assume that iteration  $k$  ends with a basic restart and let  $\varphi$  be the learned basic clause. For every  $\varphi' \in \Phi_k$ ,  $\varphi \not\vdash_c \varphi'$ .
- (viii) Assume that iteration  $k$  ends with a semantic restart and let  $\psi$  be the learned axiom. For every  $\psi' \in \Psi_k$ ,  $\chi_k(\psi) \not\vdash_i \chi_k(\psi')$ .

*Proof.* We only sketch the proof of the non-trivial points.

(iii). By Lemma 2 applied to  $\mathcal{T}_k$ .

(v). Every interpretation  $M$  generated at Step (S6) is a superset of  $r_k$ , thus after Step (S2)  $r_k$  is the minimum element of  $W$  and the root of  $\mathcal{K}(W)$ . By (iv) and Prop. 2(i),  $r_k \Vdash R_k$ . Since  $g \notin r_k$ , we get  $r_k \not\models g$ .

(vi). At Step (S4),  $w \triangleright_W \lambda$  for every  $w \in W$  and  $\lambda \in X_k$ . Since  $(X_k)^* \subseteq R_k$ , by Prop. 2(ii) we get  $r_k \Vdash X_k$ . Let  $\psi \in \Psi_k$ ; then,  $\psi$  has been learned at some iteration  $k' < k$ . Let  $(R', X', \chi')$  be the output of **Clausify**( $\psi, V$ ) at Step (S5) of iteration  $k'$ . Since  $R' \subseteq R_k$  and  $X' \subseteq X_k$ , it holds that  $r_k \Vdash R' \cup X'$ . By (P1)  $R', X' \vdash_i \psi$ , hence  $r_k \Vdash \psi$ , which proves  $r_k \Vdash \Psi_k$ .

(vii). Let  $\varphi' \in \Phi_k$ ; we show that  $\varphi \not\vdash_c \varphi'$ . Let  $\varphi = \bigwedge(A \setminus \{a\}) \rightarrow c$ ; then, there are  $w \in W$  and  $\lambda = (a \rightarrow b) \rightarrow c \in X_k$  such that  $\langle w, \lambda \rangle$  has been selected at Step (S3) and the outcome of **satProve**( $s, w \cup \{a\}, b$ ) at Step (S6) is Yes( $A$ ). Note that  $w \not\triangleright_W \lambda$ , hence  $c \notin w$ ; since  $A \subseteq w \cup \{a\}$ , we get  $w \not\models \varphi$ . On the other hand,  $w \models \varphi'$ , since  $\varphi' \in \Phi_k$  and  $\Phi_k \subseteq R_k$ . We conclude  $\varphi \not\vdash_c \varphi'$ .

(viii). Let  $\psi' \in \Psi_k$  and let  $\mathcal{K}(W)$  be the model obtained at Step (S4) of iteration  $k$ . By (iii)  $R_k, X_k \vdash_i \psi \leftrightarrow \chi_k(\psi)$  and  $R_k, X_k \vdash_i \psi' \leftrightarrow \chi_k(\psi')$ . Since  $r_k \not\models \psi$  and  $r_k \Vdash \psi'$  (indeed,  $\psi' \in \Psi_k$  and  $r_k \Vdash \Psi_k$ ) and  $r_k \Vdash R_k \cup X_k$ , we get  $r_k \not\models \chi_k(\psi)$  and  $r_k \Vdash \chi_k(\psi')$ . We conclude  $\chi_k(\psi) \not\vdash_i \chi_k(\psi')$ .  $\square$

The following proposition proves the partial correctness of **intuitRIL**:

**Proposition 3.** **intuitRIL**( $\alpha, L$ ) satisfies properties (Q1) and (Q2).

*Proof.* Let us assume that the computation ends at iteration  $k$  with output  $\Psi_\alpha$ . Then, the call to the SAT-solver at Step (S0) yields Yes( $\emptyset$ ), meaning that  $R_k \vdash_c g$ . We can build the following  $C_L$ -derivation  $\mathcal{D}$  of  $\Rightarrow \alpha$ :

$$\mathcal{D} = \frac{R_k \vdash_c g}{R_k, X_k \Rightarrow g} \text{cp}l_0 \quad \pi(\mathcal{D}) = \pi(\mathcal{T}_k) = \langle \Psi_k, \chi_k \rangle$$

$$\vdots \mathcal{T}_k$$

$$\Rightarrow \alpha$$

Note that  $\Psi_\alpha = \chi_k(\Psi_k)$ . Accordingly, by Prop. 1 we get (Q1).

Let us assume that the output is the model  $\mathcal{K}(W)$ , having root  $r$ . Then,  $\mathcal{K}(W)$  is an  $L$ -model (otherwise, Step (S4) should have forced a semantic restart). By Lemma 3(vi) we get  $r \Vdash R_0 \cup X_0$  and  $r \nVdash g$ . Since at Step (S0) we have clasified the formula  $\alpha \leftrightarrow g$ , by (P1) we get  $R_0, X_0 \vdash_i \alpha \leftrightarrow g$ , which implies  $r \Vdash \alpha \leftrightarrow g$ . We conclude that  $r \nVdash \alpha$ , hence (Q2) holds.  $\square$

It seems challenging to provide a general proof of termination, and each logic must be treated apart. We can only state some general properties about the termination of the inner loop and of consecutive basic restarts.

**Proposition 4.** (i) *The inner loop is terminating.*  
(ii) *The number of consecutive basic restarts is finite.*

*Proof.* Let us assume, by absurd, that the inner loop is not terminating. For every  $j \geq 0$ , by  $W_j$  we denote the value of  $W$  at Step (S3) of iteration  $j$  of the inner loop; note that the value of the variable  $V$  does not change during the iterations. We show that  $W_j \subset W_{j+1}$ , for every  $j \geq 0$ . At iteration  $j$ , the outcome of Step (S6) is  $\text{No}(M)$ . Thus, there are  $w \in W_j$  and  $\lambda = (a \rightarrow b) \rightarrow c \in X$  such that the pair  $\langle w, \lambda \rangle$  has been selected at Step (S3); accordingly,  $w \nVdash_{W_j} \lambda$  and  $w \cup \{a\} \subseteq M$  and  $b \notin M$ . We have  $M \notin W_j$ , otherwise we would get  $w \triangleright_{W_j} \lambda$ , a contradiction. Since  $W_{j+1} = W_j \cup \{M\}$ , this proves that  $W_j \subset W_{j+1}$ . We have shown that  $W_0 \subset W_1 \subset W_2 \dots$ . This leads to a contradiction since, for every  $j \geq 0$  and every  $w \in W_j$ ,  $w$  is a subset of  $V$  and  $V$  is finite. We conclude that the inner loop is terminating, and this proves (i).

Let us assume, by contradiction, that there is an infinite sequence of consecutive basic restarts. Then, there is  $n \geq 0$  such that, for every  $k \geq n$ , the iteration  $k$  of the main loop ends with a basic restart. Let  $\varphi_k$  be the clause learned at iteration  $k$ . Note that an iteration ending with a basic restart does not introduce new atoms, thus  $\mathcal{V}_{\varphi_k} \subseteq V_n$  for every  $k \geq n$  (where  $V_n$  is defined as in (†)). We get a contradiction, since  $V_n$  is finite and, by Lemma 3(vi), the clauses  $\varphi_k$  are pairwise non  $\sim_c$ -equivalent; this proves (ii).  $\square$

Lemma 3(vii) guarantees that the learned axioms are pairwise distinct, but this is not sufficient to prove termination since in general we cannot set a bound on the size and on the number of learned axioms. In next section we present some relevant logics where the procedure is terminating.

## 5 Termination

Let  $\text{GL} = \text{IPL} + \mathbf{lin}$  be the Gödel-Dummett logic presented in Ex. 1; we show that every call  $\text{intuitRIL}(\alpha, \text{GL})$  is terminating. To this aim, we exploit the bounding function  $\text{Ax}_{\text{GL}}(\alpha)$  presented in the mentioned example.

**Lemma 4.** *Let us consider the computation of  $\text{intuitRIL}(\alpha, \text{GL})$  and assume that at iteration  $k$  of the main loop Step (S4) is executed and that the obtained model  $\mathcal{K}(W)$  is not linear. Then, there exists  $\psi \in \text{Ax}_{\text{GL}}(\alpha)$  such that  $r_k \not\models \psi$ .*

*Proof.* Let us assume that  $\mathcal{K}(W)$  has two distinct maximal worlds  $w_1$  and  $w_2$ ; note that  $w_1 \subseteq V_k$  and  $w_2 \subseteq V_k$  (with  $V_k$  defined as in (†)). We show that:

(a)  $w_1 \cap \mathcal{V}_\alpha \neq w_2 \cap \mathcal{V}_\alpha$ .

Suppose by contradiction  $w_1 \cap \mathcal{V}_\alpha = w_2 \cap \mathcal{V}_\alpha$ ; let  $p \in V_k$  and  $\beta = \chi_k(p)$  (with  $\chi_k$  defined as in (†)). By Lemma 3(iii),  $R_k, X_k \vdash_i p \leftrightarrow \beta$ ; by Lemma 3(vi) we get  $w_1 \models p \leftrightarrow \beta$  and  $w_2 \models p \leftrightarrow \beta$ . Since  $\mathcal{V}_\beta \subseteq \mathcal{V}_\alpha$  (see Lemma 3(iii)) and we are assuming  $w_1 \cap \mathcal{V}_\alpha = w_2 \cap \mathcal{V}_\alpha$ , it holds that  $w_1 \models \beta$  iff  $w_2 \models \beta$ , thus  $w_1 \models p$  iff  $w_2 \models p$ , namely  $p \in w_1$  iff  $p \in w_2$ . Since  $p$  is any element of  $V_k$ , we get  $w_1 = w_2$ , a contradiction; this proves (a). By (a) there is  $a \in \mathcal{V}_\alpha$  such that either  $a \in w_1 \setminus w_2$  or  $a \in w_2 \setminus w_1$ . We consider the former case (the latter one is symmetric), corresponding to Case 1 in Fig. 6. We have  $w_1 \models a$  and  $w_2 \models \neg a$ ; setting  $\psi = (a \rightarrow \neg a) \vee (\neg a \rightarrow a)$ , we conclude  $r_k \not\models \psi$ .

Assume that  $\mathcal{K}(W)$  has only one maximal world; since it is not linear, there are three distinct worlds  $w_1, w_2, w_3$  as in Case 2 in Fig. 6, namely:  $w_1$  is an immediate successor of  $w_2$  and  $w_3$  (i.e., for  $j \in \{2, 3\}$ ,  $w_j < w_1$  and, if  $w_j < w$ , then  $w_1 \leq w$ ),  $w_2 \not\leq w_3$ ,  $w_3 \not\leq w_2$ . Reasoning as in (a), we get:

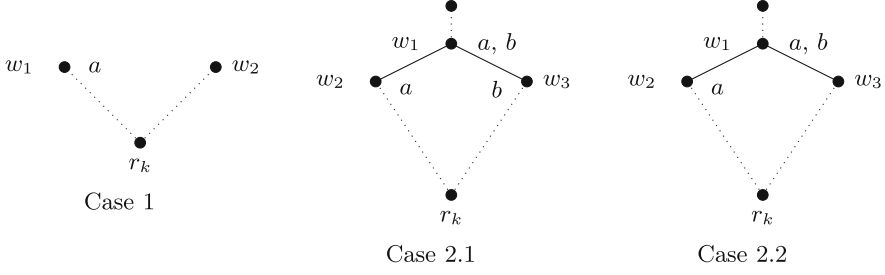
(b)  $w_2 \cap \mathcal{V}_\alpha \neq w_3 \cap \mathcal{V}_\alpha$ .      (c)  $w_2 \cap \mathcal{V}_\alpha \subset w_1 \cap \mathcal{V}_\alpha$  and  $w_3 \cap \mathcal{V}_\alpha \subset w_1 \cap \mathcal{V}_\alpha$ .

By (b) there is  $a \in \mathcal{V}_\alpha$  such that either  $a \in w_2 \setminus w_3$  or  $a \in w_3 \setminus w_2$ . Let us consider the former case (the latter one is symmetric). By (c), there is  $b \in \mathcal{V}_\alpha$  such that  $b \in w_1 \setminus w_2$ . If  $b \in w_3$  (Case 2.1 in Fig. 6), we get  $a \in w_2$ ,  $b \notin w_2$ ,  $a \notin w_3$ ,  $b \in w_3$ . Setting  $\psi = (a \rightarrow b) \vee (b \rightarrow a)$ , we conclude  $r_k \not\models \psi$ . Finally, let us assume  $b \notin w_3$  (Case 2.2). We have  $\{a, b\} \subseteq w_1$ ,  $a \in w_2$ ,  $b \notin w_2$ ,  $a \notin w_3$  and  $b \notin w_3$ . It is easy to check that  $w_3 \models a \rightarrow b$  (recall that  $w_3 < w$  implies  $w_1 \leq w$ ), thus  $w_3 \not\models (a \rightarrow b) \rightarrow a$ . On the other hand  $w_2 \not\models a \rightarrow (a \rightarrow b)$ . Setting  $\psi = (a \rightarrow (a \rightarrow b)) \vee ((a \rightarrow b) \rightarrow a)$ , we get  $r_k \not\models \psi$ .  $\square$

We exploit Lemma 4 to implement Step (S4). If  $\mathcal{K}(W)$  is linear, then  $\mathcal{K}(W)$  is a GL-model and we are done. Otherwise, the proof of Lemma 4 hints an effective method to select an instance  $\psi$  of  $\mathbf{lin}$  from  $\text{Ax}_{\text{GL}}(\alpha)$ .

**Proposition 5.** *The computation of  $\text{intuitRIL}(\alpha, \text{GL})$  is terminating.*

*Proof.* Assume that  $\text{intuitRIL}(\alpha, \text{GL})$  is not terminating. Since the number of iterations of the inner loop and of the consecutive basic restarts is finite (see Prop. 4), Step (S4) must be executed infinitely many times. This leads to a contradiction, since the axioms selected at Step (S4) are pairwise distinct (see Lemma 3(vii)) and such axioms are chosen from the finite set  $\text{Ax}_{\text{GL}}(\alpha)$ .  $\square$



**Fig. 6.** Proof of Lemma 4, case analysis.

As a corollary, we get that  $\text{Ax}_{\text{GL}}(\alpha)$  is a bounding function for GL:

**Proposition 6.** *If  $\alpha$  is GL-valid, there is  $\Psi_\alpha \subseteq \text{Ax}_{\text{GL}}(\alpha)$  such that  $\Psi_\alpha \vdash_i \alpha$ .*

Other proof-search strategies for GL are discussed in [10, 14]. This technique can be extended to other notable intermediate logics. Among these, we recall the logics  $\text{GL}_n$  (Gödel Logic of depth  $n$ ), obtained by adding to GL the axioms  $\mathbf{bd}_n$  (bounded depth) where:  $\mathbf{bd}_0 = a_0 \vee \neg a_0$ ,  $\mathbf{bd}_{n+1} = a_{n+1} \vee (a_{n+1} \rightarrow \mathbf{bd}_n)$ . Semantically,  $\text{GL}_n$  is the logic characterized by linear frames having depth at most  $n$ . We are not able to prove termination for the logics  $\text{IPL} + \mathbf{bd}_n$ , but we can implement the following terminating strategy for  $\text{GL}_n$ . Let  $\mathcal{K}(W)$  be the model obtained at Step (S4) of the computation of  $\text{intuitRIL}(\alpha, \text{GL}_n)$ :

- If  $\mathcal{K}(W)$  is not linear, we select the axiom  $\psi$  from  $\text{Ax}_{\text{GL}}(\alpha)$ .
- Otherwise, assume that  $\mathcal{K}(W)$  is linear but not a  $\text{GL}_n$ -model. Then,  $\mathcal{K}(W)$  contains a chain of worlds  $w_0 \subset w_1 \subset \dots \subset w_{n+1}$ . The crucial point is that  $w_{j+1} \setminus w_j$  contains at least a propositional variable from  $\mathcal{V}_\alpha$ , for every  $0 \leq j \leq n$ . Thus, we can choose a proper renaming of  $\mathbf{bd}_n$  as  $\psi$ .

Another terminating logic is the Jankov Logic (see Ex. 3); actually, also in this case the learned axiom can be chosen by renaming the **wem** axiom. In general, all the logics  $\text{BTW}_n$  (Bounded Top Width, at most  $n$  maximal worlds, see [2]) are terminating. An intriguing case is Scott Logic ST [2]: even though the class of ST-frames is not first-order definable, we can implement a learning procedure for ST-axioms arguing as in [7] (see Sec. 2.5.2). Some of the mentioned logics have been implemented in *intuitRIL*<sup>1</sup>.

One may wonder whether this method can be applied to other non-classical logics or to fragments of predicate logics (these issues have been already raised in the seminal paper [4]). A significant work in this direction is [11], where the procedure has been applied to some modal logics. However, the main difference with the original approach is that it is not possible to use a single SAT-solver, but one needs a supply of SAT-solvers. This is primarily due to the fact that forcing relation of modal Kripke models is not persistent; thus worlds are loosely related and must be handled by independent solvers.

<sup>1</sup> Available at <https://github.com/cfiorentini/intuitRIL>.

## References

1. Avellone, A., Moscato, U., Miglioli, P., Ornaghi, M.: Generalized tableau systems for intermediate propositional logics. In: Galmiche, D. (ed.) *TABLEAUX 1997*. LNCS, vol. 1227, pp. 43–61. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0027404>
2. Chagrov, A.V., Zakharyashev, M.: *Modal Logic*, Oxford Logic Guides, vol. 35. Oxford University Press (1997)
3. Ciabattoni, A., Lang, T., Ramanayake, R.: Bounded-analytic sequent calculi and embeddings for hypersequent logics. *J. Symb. Log.* **86**(2), 635–668 (2021)
4. Claessen, K., Rosén, D.: SAT modulo intuitionistic implications. In: Davis, M., Fehner, A., McIver, A., Voronkov, A. (eds.) *LPAR 2015*. LNCS, vol. 9450, pp. 622–637. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48899-7\\_43](https://doi.org/10.1007/978-3-662-48899-7_43)
5. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. *J. Symb. Log.* **57**(3), 795–807 (1992)
6. Ferrari, M., Fiorentini, C., Fiorino, G.: FCUBE: an efficient prover for intuitionistic propositional logic. In: Fermüller, C.G., Voronkov, A. (eds.) *LPAR 2010*. LNCS, vol. 6397, pp. 294–301. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-16242-8\\_21](https://doi.org/10.1007/978-3-642-16242-8_21)
7. Fiorentini, C.: Kripke completeness for intermediate logics. Ph.D. thesis, Università degli Studi di Milano (2000)
8. Fiorentini, C.: Efficient SAT-based proof search in intuitionistic propositional logic. In: Platzer, A., Sutcliffe, G. (eds.) *CADE 2021*. LNCS (LNAI), vol. 12699, pp. 217–233. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_13](https://doi.org/10.1007/978-3-030-79876-5_13)
9. Fiorentini, C., Goré, R., Graham-Lengrand, S.: A proof-theoretic perspective on SMT-solving for intuitionistic propositional logic. In: Cerrito, S., Popescu, A. (eds.) *TABLEAUX 2019*. LNCS (LNAI), vol. 11714, pp. 111–129. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-29026-9\\_7](https://doi.org/10.1007/978-3-030-29026-9_7)
10. Fiorino, G.: Terminating calculi for propositional dummett logic with subformula property. *J. Autom. Reason.* **52**(1), 67–97 (2013). <https://doi.org/10.1007/s10817-013-9276-7>
11. Goré, R., Kikkert, C.: CEGAR-tableaux: improved modal satisfiability via modal clause-learning and SAT. In: Das, A., Negri, S. (eds.) *TABLEAUX 2021*. LNCS (LNAI), vol. 12842, pp. 74–91. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-86059-2\\_5](https://doi.org/10.1007/978-3-030-86059-2_5)
12. Goré, R., Thomson, J., Wu, J.: A history-based theorem prover for intuitionistic propositional logic using global caching: IntHistGC system description. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) *IJCAR 2014*. LNCS (LNAI), vol. 8562, pp. 262–268. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08587-6\\_19](https://doi.org/10.1007/978-3-319-08587-6_19)
13. Jankov, V.: The calculus of the weak “law of excluded middle.”. *Math. USSR* **8**, 648–650 (1968)
14. Larchey-Wendling, D.: Gödel-dummett counter-models through matrix computation. *Electron. Notes Theory Comput. Sci.* **125**(3), 137–148 (2005)
15. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. *ACM Trans. Comput. Log.* **2**(4), 526–541 (2001)
16. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *J. ACM* **53**(6), 937–977 (2006)



17. Schmidt, R.A., Tishkovsky, D.: Automated synthesis of tableau calculi. *Log. Methods Comput. Sci.* **7**(2) (2011)
18. Troelstra, A.S., Schwichtenberg, H.: *Basic Proof Theory*, Cambridge Tracts in Theoretical Computer Science, vol. 43, 2nd edn. Cambridge University Press, Cambridge (2000)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Clause Redundancy and Preprocessing in Maximum Satisfiability

Hannes Ihalainen, Jeremias Berg<sup>(✉)</sup> , and Matti Järvisalo 

HIIT, Department of Computer Science, University of Helsinki, Helsinki, Finland

`hannes.ihalainen@helsinki.fi` `jeremias.berg@helsinki.fi`

`matti.jarvisalo@helsinki.fi`

**Abstract.** The study of clause redundancy in Boolean satisfiability (SAT) has proven significant in various terms, from fundamental insights into preprocessing and inprocessing to the development of practical proof checkers and new types of strong proof systems. We study liftings of the recently-proposed notion of propagation redundancy—based on a semantic implication relationship between formulas—in the context of maximum satisfiability (MaxSAT), where of interest are reasoning techniques that preserve optimal cost (in contrast to preserving satisfiability in the realm of SAT). We establish that the strongest MaxSAT-lifting of propagation redundancy allows for changing in a controlled way the set of minimal correction sets in MaxSAT. This ability is key in succinctly expressing MaxSAT reasoning techniques and allows for obtaining correctness proofs in a uniform way for MaxSAT reasoning techniques very generally. Bridging theory to practice, we also provide a new MaxSAT preprocessor incorporating such extended techniques, and show through experiments its wide applicability in improving the performance of modern MaxSAT solvers.

**Keywords:** Maximum satisfiability · Clause redundancy ·  
Propagation redundancy · Preprocessing

## 1 Introduction

Building heavily on the success of Boolean satisfiability (SAT) solving [13], maximum satisfiability (MaxSAT) as the optimization extension of SAT constitutes a viable approach to solving real-world NP-hard optimization problems [6, 35]. In the context of SAT, the study of fundamental aspects of clause redundancy [20, 21, 23, 28, 29, 31, 32] has proven central for developing novel types of preprocessing and inprocessing-style solving techniques [24, 29] as well as in enabling efficient proof checkers [7, 15, 16, 18, 19, 41, 42] via succinct representation of most practical SAT solving techniques. Furthermore, clause redundancy notions have

---

Work financially supported by Academy of Finland under grants 322869, 328718 and 342145. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 75–94, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_6](https://doi.org/10.1007/978-3-031-10769-6_6)

been shown to give rise to very powerful proof systems, going far beyond resolution [22, 23, 30]. In contrast to viewing clause redundancy through the lens of logical entailment, the redundancy criteria developed in this line of work are based on a semantic implication relationship between formulas, making them desirably efficient to decide and at the same time are guaranteed to merely preserve satisfiability rather than logical equivalence.

The focus of this work is the study of clause redundancy in the context of MaxSAT through lifting recently-proposed variants of the notion of *propagation redundancy* [23] based on a semantic implication relationship between formulas from the realm of SAT. The study of such liftings is motivated from several perspectives. Firstly, earlier it has been shown that a natural MaxSAT-lifting called SRAT [10] of the redundancy notion of the notion of *resolution asymmetric tautologies* (RAT) [29] allows for establishing the general correctness of MaxSAT-liftings of typical preprocessing techniques in SAT solving [14], alleviating the need for correctness proofs for individual preprocessing techniques [8]. However, the need for preserving the *optimal cost* in MaxSAT—as a natural counterpart for preserving satisfiability in SAT—allows for developing MaxSAT-centric preprocessing and solving techniques which cannot be expressed through SRAT [2, 11]. Capturing more generally such cost-aware techniques requires developing more expressive notions of clause redundancy. Secondly, due to the fundamental connections between solutions and so-called minimal corrections sets (MCSes) of MaxSAT instances [8, 25], analyzing the effect of clauses that are redundant in terms of expressive notions of redundancy on the MCSes of MaxSAT instances can provide further understanding on the relationship between the different notions and their fundamental impact on the solutions of MaxSAT instances. Furthermore, in analogy with SAT, more expressive redundancy notions may prove fruitful for developing further practical preprocessing and solving techniques for MaxSAT.

Our main contributions are the following. We propose natural liftings of the three recently-proposed variants PR, LPR and SPR of propagation redundancy in the context of SAT to MaxSAT. We provide a complete characterization of the relative expressiveness of the lifted notions CPR, CLPR and CSPR (C standing for cost for short) and of their impact on the set of MCSes in MaxSAT instances. In particular, while removing or adding clauses redundant in terms of CSPR and CLPR (the latter shown to be equivalent with SRAT) do not influence the set of MCSes underlying MaxSAT instances, CPR can in fact have an influence on MCSes. In terms of solutions, this result implies that CSPR or CLPR clauses can not remove minimal (in terms of sum-of-weights of falsified soft clauses) solutions of MaxSAT instances, while CPR clauses can.

The—theoretically greater—effect that CPR clauses have on the solutions of MaxSAT instances is key for succinctly expressing further MaxSAT reasoning techniques via CPR and allows for obtaining correctness proofs in a uniform way for MaxSAT reasoning techniques very generally; we give concrete examples of how CPR captures techniques not in the reach of SRAT. Bridging to practical preprocessing in MaxSAT, we also provide a new MaxSAT preprocessor

extended with such techniques. Finally, we provide large-scale empirical evidence on the positive impact of the preprocessor on the runtimes of various modern MaxSAT solvers, covering both complete and incomplete approaches, suggesting that extensive preprocessing going beyond the scope of SRAT appears beneficial to integrate for speeding up modern MaxSAT solvers.

An extended version of this paper, with formal proofs missing from this version, is available via the authors' homepages.

## 2 Preliminaries

**SAT.** For a Boolean variable  $x$  there are two literals, the positive  $x$  and the negative  $\neg x$ , with  $\neg\neg l = l$  for a literal  $l$ . A clause  $C$  is a set (disjunction) of literals and a CNF formula  $F$  a set (conjunction) of clauses. We assume that all clauses are non-tautological, i.e., do not contain both a literal and its negation. The set  $\text{var}(C) = \{x \mid x \in C \text{ or } \neg x \in C\}$  consists of the variables of the literals in  $C$ . The set of variables and literals, respectively, of a formula are  $\text{var}(F) = \bigcup_{C \in F} \text{var}(C)$  and  $\text{lit}(F) = \bigcup_{C \in F} C$ , respectively. For a set  $L$  of literals, the set  $\neg L = \{\neg l \mid l \in L\}$  consists of the negations of the literals in  $L$ .

A (*truth*) *assignment*  $\tau$  is a set of literals for which  $x \notin \tau$  or  $\neg x \notin \tau$  for any variable  $x$ . For a literal  $l$  we denote  $l \in \tau$  by  $\tau(l) = 1$  and  $\neg l \in \tau$  by  $\tau(l) = 0$  or  $\tau(\neg l) = 1$  as convenient, and say that  $\tau$  assigns  $l$  the value 1 and 0, respectively. The set  $\text{var}(\tau) = \{x \mid x \in \tau \text{ or } \neg x \in \tau\}$  is the range of  $\tau$ , i.e., it consists of the variables  $\tau$  assigns a value for. For a set  $L$  of literals and an assignment  $\tau$ , the assignment  $\tau_L = (\tau \setminus \neg L) \cup L$  is obtained from  $\tau$  by setting  $\tau_L(l) = 1$  for all  $l \in L$  and  $\tau_L(l) = \tau(l)$  for all  $l \notin L$  assigned by  $\tau$ . For a literal  $l$ ,  $\tau_l$  stands for  $\tau_{\{l\}}$ . An assignment  $\tau$  satisfies a clause  $C$  ( $\tau(C) = 1$ ) if  $\tau \cap C \neq \emptyset$  or equivalently if  $\tau(l) = 1$  for some  $l \in C$ , and a CNF formula  $F$  ( $\tau(F) = 1$ ) if it satisfies each clause  $C \in F$ . A CNF formula is satisfiable if there is an assignment that satisfies it, and otherwise unsatisfiable. The empty formula  $\top$  is satisfied by any truth assignment and the empty clause  $\perp$  is unsatisfiable. The Boolean satisfiability problem (SAT) asks to decide whether a given CNF formula  $F$  is satisfiable.

Given two CNF formulas  $F_1$  and  $F_2$ ,  $F_1$  entails  $F_2$  ( $F_1 \models F_2$ ) if any assignment  $\tau$  that satisfies  $F_1$  and only assigns variables of  $F_1$  (i.e. for which  $\text{var}(\tau) \subset \text{var}(F_1)$ ) can be extended into an assignment  $\tau^2 \supset \tau$  that satisfies  $F_2$ . The formulas are equisatisfiable if  $F_1$  is satisfiable iff  $F_2$  is. An assignment  $\tau$  is complete for a CNF formula  $F$  if  $\text{var}(F) \subset \text{var}(\tau)$ , and otherwise partial for  $F$ . The restriction  $F|_{\tau}$  of  $F$  wrt a partial assignment  $\tau$  is a CNF formula obtained by (i) removing from  $F$  all clauses that are satisfied by  $\tau$  and (ii) removing from the remaining clauses of  $F$  literals  $l$  for which  $\tau(l) = 0$ . Applying unit propagation on  $F$  refers to iteratively restricting  $F$  by  $\tau = \{l\}$  for a unit clause (clause with a single literal)  $(l) \in F$  until the resulting (unique) formula, denoted by  $\text{UP}(F)$ , contains no unit clauses or some clause in  $F$  becomes empty. We say that unit propagation on  $F$  derives a conflict if  $\text{UP}(F)$  contains the empty clause. The formula  $F_1$  implies  $F_2$  under unit propagation ( $F_1 \vdash_1 F_2$ ) if, for each  $C \in F_2$ ,

unit propagation derives a conflict in  $F_1 \wedge \{(\neg l) \mid l \in C\}$ . Note that  $F_1 \vdash_1 F_2$  implies  $F_1 \models F_2$ , but not vice versa in general.

**Maximum Satisfiability.** An instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{F}_S, w)$  of (weighted partial) maximum satisfiability (MaxSAT for short) consists of two CNF formulas, the hard clauses  $\mathcal{F}_H$  and the soft clauses  $\mathcal{F}_S$ , and a weight function  $w: \mathcal{F}_S \rightarrow \mathbb{N}$  that assigns a positive weight to each soft clause.

Without loss of generality, we assume that every soft clause  $C \in \mathcal{F}_S$  is unit<sup>1</sup>. The set of *blocking* literals  $\mathcal{B}(\mathcal{F}) = \{l \mid (\neg l) \in \mathcal{F}_S\}$  consists of the literals  $l$  the negation of which occurs in  $\mathcal{F}_S$ . The weight function  $w$  is extended to blocking literals by  $w(l) = w((\neg l))$ . Without loss of generality, we also assume that  $l \in \text{lit}(\mathcal{F}_H)$  for all  $l \in \mathcal{B}(\mathcal{F})$ <sup>2</sup>. Instead of using the definition of MaxSAT in terms of hard and soft clauses, we will from now on view a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  as a set  $\mathcal{F}_H$  of hard clauses, a set  $\mathcal{B}(\mathcal{F})$  of blocking literals and a weight function  $w: \mathcal{B}(\mathcal{F}) \rightarrow \mathbb{N}$ .

Any complete assignment  $\tau$  over  $\text{var}(\mathcal{F}_H)$  that satisfies  $\mathcal{F}_H$  is a solution to  $\mathcal{F}$ . The cost  $\text{COST}(\mathcal{F}, \tau) = \sum_{l \in \mathcal{B}(\mathcal{F})} \tau(l)w(l)$  of a solution  $\tau$  is the sum of weights of blocking literals it assigns to 1<sup>3</sup>. The cost of a complete assignment  $\tau$  that does not satisfy  $\mathcal{F}_H$  is defined as  $\infty$ . The cost of a partial assignment  $\tau$  over  $\text{var}(\mathcal{F}_H)$  is defined as the cost of smallest-cost assignments that are extensions of  $\tau$ . A solution  $\tau^\circ$  is optimal if  $\text{COST}(\mathcal{F}, \tau^\circ) \leq \text{COST}(\mathcal{F}, \tau)$  holds for all solutions  $\tau$  of  $\mathcal{F}$ . The cost of the optimal solutions of a MaxSAT instance is denoted by  $\text{COST}(\mathcal{F})$ , with  $\text{COST}(\mathcal{F}) = \infty$  iff  $\mathcal{F}_H$  is unsatisfiable. In MaxSAT the task is to find an optimal solution to a given MaxSAT instance.

*Example 1.* Let  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  be a MaxSAT instance with  $\mathcal{F}_H = \{(x \vee b_1), (\neg x \vee b_2), (y \vee b_3 \vee b_4), (z \vee \neg y \vee b_4), (\neg z)\}$ ,  $\mathcal{B}(\mathcal{F}) = \{b_1, b_2, b_3, b_4\}$  having  $w(b_1) = w(b_4) = 1$ ,  $w(b_2) = 2$  and  $w(b_3) = 8$ . The assignment  $\tau = \{b_1, b_4, \neg b_2, \neg b_3, \neg x, \neg z, y\}$  is an example of an optimal solution of  $\mathcal{F}$  and has  $\text{COST}(\mathcal{F}, \tau) = \text{COST}(\mathcal{F}) = 2$ .

With a slight abuse of notation, we denote by  $\mathcal{F} \wedge C = (\mathcal{F}_H \cup \{C\}, \mathcal{B}(\mathcal{F} \wedge C), w)$  the MaxSAT instance obtained by adding a clause  $C$  to an instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$ . Adding clauses may introduce new blocking literals but not change the weights of already existing ones, i.e.,  $\mathcal{B}(\mathcal{F}) \subset \mathcal{B}(\mathcal{F} \wedge C)$  and  $w^{\mathcal{F}}(l) = w^{\mathcal{F} \wedge C}(l)$  for all  $l \in \mathcal{B}(\mathcal{F})$ .

**Correction Sets.** For a MaxSAT instance  $\mathcal{F}$ , a subset  $\text{cs} \subset \mathcal{B}(\mathcal{F})$  is a minimal correction set (MCS) of  $\mathcal{F}$  if (i)  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus \text{cs}} (\neg l)$  is satisfiable and (ii)  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus \text{cs}_s} (\neg l)$  is unsatisfiable for every  $\text{cs}_s \subsetneq \text{cs}$ . In words,  $\text{cs}$  is an MCS if it

<sup>1</sup> A soft clause  $C$  can be replaced by the hard clause  $C \vee x$  and soft clause  $(\neg x)$ , where  $x$  is a variable not in  $\text{var}(\mathcal{F}_H \wedge \mathcal{F}_S)$ , without affecting the costs of solutions.

<sup>2</sup> Otherwise the instance can be simplified by unit propagating  $\neg l$  without changing the costs of solutions. As a consequence, any complete assignment for  $\mathcal{F}_H$  will be complete for  $\mathcal{F}_H \wedge \mathcal{F}_S$  as well.

<sup>3</sup> This is equivalent to the sum of weights of soft clauses not satisfied by  $\tau$ .

is a subset-minimal set of blocking literals that is included in some solution  $\tau$  of  $\mathcal{F}$ .<sup>4</sup> We denote the set of MCSes of  $\mathcal{F}$  by  $\text{mcs}(\mathcal{F})$ .

There is a tight connection between the MCSes and solutions of MaxSAT instances. Given an optimal solution  $\tau^o$  of a MaxSAT instance  $\mathcal{F}$ , the set  $\tau^o \cap \mathcal{B}(\mathcal{F})$  is an MCS of  $\mathcal{F}$ . In the other direction, for any  $\text{cs} \in \text{mcs}(\mathcal{F})$ , there is a (not necessary optimal) solution  $\tau^{\text{cs}}$  such that  $\text{cs} = \mathcal{B}(\mathcal{F}) \cap \tau^{\text{cs}}$  and  $\text{COST}(\mathcal{F}, \tau^{\text{cs}}) = \sum_{l \in \text{cs}} w(l)$ .

*Example 2.* Consider the instance  $\mathcal{F}$  from Example 1. The set  $\{b_1, b_4\} \in \text{mcs}(\mathcal{F})$  is an MCS of  $\mathcal{F}$  that corresponds to the optimal solution  $\tau$  described in Example 1. The set  $\{b_2, b_3\} \in \text{mcs}(\mathcal{F})$  is another example of an MCS that instead corresponds to the solution  $\tau_2 = \{b_2, b_3, \neg b_1, \neg b_4, x, \neg z, \neg y\}$  for which  $\text{COST}(\mathcal{F}, \tau) = 10$ .

### 3 Propagation Redundancy in MaxSAT

We extend recent work [23] on characterizing redundant clauses using semantic implication in the context of SAT to MaxSAT. In particular, we provide natural counterparts for several recently-proposed strong notions of redundancy in SAT to the context of MaxSAT and analyze the relationships between them.

In the context of SAT, the most general notion of clause redundancy is seemingly simple: a clause  $C$  is redundant for a formula  $F$  if it does not affect its satisfiability, i.e., clause  $C$  is redundant wrt a CNF formula  $F$  if  $F$  and  $F \wedge \{C\}$  are equisatisfiable [20, 29]. This allows for the set of satisfying assignments to change, and does not require preserving logical equivalence; we are only interested in satisfiability.

A natural counterpart for this general view in MaxSAT is that the *cost* of optimal solutions (rather than the set of optimal solutions) should be preserved.

**Definition 1.** *A clause  $C$  is redundant wrt a MaxSAT instance  $\mathcal{F}$  if  $\text{COST}(\mathcal{F}) = \text{COST}(\mathcal{F} \wedge C)$ .*

This coincides with the counterpart in SAT whenever  $\mathcal{B}(\mathcal{F}) = \emptyset$ , since then the cost of a MaxSAT instance  $\mathcal{F}$  is either 0 (if  $\mathcal{F}_H$  is satisfiable) or  $\infty$  (if  $\mathcal{F}_H$  is unsatisfiable). Unless explicitly specified, we will use the term “redundant” to refer to Definition 1.

Following [23], we say that a clause  $C$  *blocks* the assignment  $\neg C$  (and all assignments  $\tau$  for which  $\neg C \subset \tau$ ). As shown in the context of SAT [23], a clause  $C$  is redundant (in the equisatisfiability sense) for a CNF formula  $F$  if  $C$  does not block all of its satisfying assignments. The counterpart that arises in the context of MaxSAT from Definition 1 is that the cost of at least one of the solutions not blocked by  $C$  is no greater than the cost of  $\neg C$ .

**Proposition 1.** *A clause  $C$  is redundant wrt a MaxSAT instance  $\mathcal{F}$  if and only if there is an assignment  $\tau$  for which  $\text{COST}(\mathcal{F} \wedge C, \tau) = \text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \neg C)$ .*

<sup>4</sup> This is equivalent to a subset-minimal set of soft clauses falsified by  $\tau$ .

The equality  $\text{COST}(\mathcal{F} \wedge C, \tau) = \text{COST}(\mathcal{F}, \tau)$  of Proposition 1 is necessary, as witnessed by the following example.

*Example 3.* Consider the MaxSAT instance  $\mathcal{F}$  detailed in Example 1, the clause  $C = (b_5)$  with  $b_5 \in \mathcal{B}(\mathcal{F} \wedge C)$  and the assignment  $\tau = \{b_5\}$ . Then  $2 = \text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \neg C) = 2$  but  $C$  is not redundant since  $\text{COST}(\mathcal{F} \wedge C) = 2 + w^{\mathcal{F} \wedge C}(b_5) > 2 = \text{COST}(\mathcal{F})$ .

Proposition 1 provides a sufficient condition for a clause  $C$  being redundant. Further requirements on the assignment  $\tau$  can be imposed without loss of generality.

**Theorem 1.** *A non-empty clause  $C$  is redundant wrt a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  if and only if there is an assignment  $\tau$  such that*

- (i)  $\tau(C) = 1$ ,
- (ii)  $\mathcal{F}_H|_{\neg C} \models \mathcal{F}_H|_{\tau}$  and
- (iii)  $\text{COST}(\mathcal{F} \wedge C, \tau) = \text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \neg C)$ .

As we will see later, a reason for including two additional conditions in Theorem 1 is to allow defining different restrictions of redundancy notions, some of which allow for efficiently identifying redundant clauses.

*Example 4.* Consider the instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  detailed in Example 1, a clause  $C = (\neg x \vee b_5)$  for a  $b_5 \in \mathcal{B}(\mathcal{F} \wedge C)$  and an assignment  $\tau = \{\neg x, b_1\}$ . Then:  $\tau(C) = 1$ ,  $\{(b_2), (y \vee b_3 \vee b_4), (z \vee \neg y \vee b_4), (\neg z)\} = \mathcal{F}_H|_{\neg C} \models \mathcal{F}_H|_{\tau} = \{(y \vee b_3 \vee b_4), (z \vee \neg y \vee b_4), (\neg z)\}$ , and  $2 = \text{COST}(\mathcal{F} \wedge C, \tau) = \text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \neg C) = 3$ . We conclude that  $C$  is redundant.

In the context of SAT, imposing restrictions on the entailment operator and the set of assignments has been shown to give rise to several interesting redundancy notions which hold promise of practical applicability. These include three variants (LPR, SPR, and PR) of so-called (literal/set) propagation redundancy [23]. For completeness we restate the definitions of these three notions. A clause  $C$  is LPR wrt a CNF formula  $F$  if there is a literal  $l \in C$  for which  $F|_{\neg C} \vdash_1 F|_{(\neg C)_l}$ , SPR if the same holds for a subset  $L \subset C$ , and PR if there exists an assignment  $\tau$  that satisfies  $C$  and for which  $F|_{\neg C} \vdash_1 F|_{\tau}$ . With the help of Theorem 1, we obtain counterparts for these notions in the context of MaxSAT.

**Definition 2.** *With respect to an instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$ , a clause  $C$  is*

- **cost literal propagation redundant (CLPR)** (on  $l$ ) *there is a literal  $l \in C$  for which either (i)  $\perp \in \text{UP}(\mathcal{F}_H|_{\neg C})$  or (ii)  $l \notin \mathcal{B}(\mathcal{F} \wedge C)$  and  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{(\neg C)_l}$ ;*
- **cost set propagation redundant (CSPR)** (on  $L$ ) *if there is a set  $L \subset C \setminus \mathcal{B}(\mathcal{F} \wedge C)$  of literals for which  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{(\neg C)_L}$ ; and*
- **cost propagation redundant (CPR)** *if there is an assignment  $\tau$  such that (i)  $\tau(C) = 1$ , (ii)  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{\tau}$  and (iii)  $\text{COST}(\mathcal{F} \wedge C, \tau) = \text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \neg C)$ .*

*Example 5.* Consider again  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  from Example 1. The clause  $D = (b_1 \vee b_2)$  is CLPR wrt  $\mathcal{F}$  since  $\perp \in \text{UP}(\mathcal{F}_H|_{\neg D})$  as  $\{(x), (\neg x)\} \subset \mathcal{F}_H|_{\neg D}$ . As for the redundant clause  $C$  and assignment  $\tau$  detailed in Example 3, we have that  $C$  is CPR, since  $\mathcal{F}_H|_{\tau} \subset \mathcal{F}_H|_{\neg C}$  which implies  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{\tau}$ .

We begin the analysis of the relationship between these redundancy notions by showing that CSPR (and by extension CLPR) clauses also satisfy the MaxSAT-centric condition (iii) of Theorem 1. Assume that  $C$  is CSPR wrt a instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  on the set  $L$ .

**Lemma 1.** *Let  $\tau \supset \neg C$  be a solution of  $\mathcal{F}$ . Then,  $\text{COST}(\mathcal{F}, \tau) \geq \text{COST}(\mathcal{F}, \tau_L)$ .*

The following corollary of Lemma 1 establishes that CSPR and CLPR clauses are redundant according to Definition 1.

**Corollary 1.**  $\text{COST}(\mathcal{F} \wedge C, (\neg C)_L) = \text{COST}(\mathcal{F}, (\neg C)_L) \leq \text{COST}(\mathcal{F}, \neg C)$ .

The fact that CPR clauses are redundant follows trivially from the fact that  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{\tau}$  implies  $\mathcal{F}_H|_{\neg C} \models \mathcal{F}_H|_{\tau}$ . However, given a solution  $\omega$  that does not satisfy a CPR clause  $C$ , the next example demonstrates that the assignment  $\omega_{\tau}$  need not have a cost lower than  $\omega$ . Stated in another way, the example demonstrates that an observation similar to Lemma 1 does not hold for CPR clauses in general.

*Example 6.* Consider a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  having  $\mathcal{F}_H = \{(x \vee b_1), (\neg x, b_2)\}$ ,  $\mathcal{B}(\mathcal{F}) = \{b_1, b_2\}$  and  $w(b_1) = w(b_2) = 1$ . The clause  $C = (x)$  is CPR wrt  $\mathcal{F}$ , the assignment  $\tau = \{x, b_2\}$  satisfies the three conditions of Definition 2. Now  $\delta = \{\neg x, b_1\}$  is a solution of  $\mathcal{F}$  that does not satisfy  $C$  for which  $\delta_{\tau} = \{x, b_1, b_2\}$  and  $1 = \text{COST}(\mathcal{F}, \delta) < 2 = \text{COST}(\mathcal{F}, \delta_{\tau})$ .

Similarly as in the context of SAT, verifying that a clause is CSPR (and by extension CLPR) can be done efficiently. However, in contrast to SAT, we conjecture that verifying that a clause is CPR can not in the general case be done efficiently, *even if the assignment  $\tau$  is given*. While we will not go into detail on the complexity of identifying CPR clauses, the following proposition gives some support for our conjecture.

**Proposition 2.** *Let  $\mathcal{F}$  be an instance and  $k \in \mathbb{N}$ . There is another instance  $\mathcal{F}^M$ , a clause  $C$ , and an assignment  $\tau$  such that  $C$  is CPR wrt  $\mathcal{F}^M$  if and only if  $\text{COST}(\mathcal{F}) \geq k$ .*

As deciding if  $\text{COST}(\mathcal{F}) \geq k$  is NP-complete in the general case, Proposition 2 suggests that it may not be possible to decide in polynomial time if an assignment  $\tau$  satisfies the three conditions of Definition 2 unless  $P=NP$ . This is in contrast to SAT, where verifying propagation redundancy can be done in polynomial time if the assignment  $\tau$  is given, but is NP-complete if not [24].

The following observations establish a more precise relationship between the redundancy notions. For the following, let  $\text{RED}(\mathcal{F})$  denote the set of clauses that are redundant wrt a MaxSAT instance  $\mathcal{F}$  according to Definition 1. Analogously, the sets  $\text{CPR}(\mathcal{F})$ ,  $\text{CSPR}(\mathcal{F})$  and  $\text{CLPR}(\mathcal{F})$  consist of the clauses that are CPR, CSPR and CLPR wrt  $\mathcal{F}$ , respectively.



**Observation 1**  $\text{CLPR}(\mathcal{F}) \subset \text{CSPR}(\mathcal{F}) \subset \text{CPR}(\mathcal{F}) \subset \text{RED}(\mathcal{F})$  holds for any MaxSAT instance  $\mathcal{F}$ .

**Observation 2** There are MaxSAT instances  $\mathcal{F}_1, \mathcal{F}_2$  and  $\mathcal{F}_3$  for which  $\text{CLPR}(\mathcal{F}_1) \subsetneq \text{CSPR}(\mathcal{F}_1)$ ,  $\text{CSPR}(\mathcal{F}_2) \subsetneq \text{CPR}(\mathcal{F}_2)$  and  $\text{CPR}(\mathcal{F}_3) \subsetneq \text{RED}(\mathcal{F}_3)$ .

The proofs of Observations 1 and 2 follow directly from known results in the context of SAT [23] by noting that any CNF formula can be viewed as an instance of MaxSAT without blocking literals.

For a MaxSAT-centric observation on the relationship between the redundancy notions, we note that the concept of redundancy and CPR coincide for any MaxSAT instance that has solutions.

**Observation 3**  $\text{CPR}(\mathcal{F}) = \text{RED}(\mathcal{F})$  holds for any MaxSAT instance  $\mathcal{F}$  with  $\text{COST}(\mathcal{F}) < \infty$ .

We note that a result similar to Observation 3 could be formulated in the context of SAT. The SAT-counterpart would state that the concept of redundancy (in the equisatisfiability sense) coincides with the concept of propagation redundancy for SAT solving (defined e.g. in [23]) for *satisfiable* CNF formulas. However, assuming that a CNF formula is satisfiable is very restrictive in the context of SAT. In contrast, it is natural to assume that a MaxSAT instance admits solutions.

We end this section with a simple observation: adding a redundant clause  $C$  to a MaxSAT instance  $\mathcal{F}$  preserves not only optimal cost, but optimal solutions of  $\mathcal{F} \wedge C$  are also optimal solutions of  $\mathcal{F}$ . However, the converse need not hold; an instance  $\mathcal{F}$  might have optimal solutions that do not satisfy  $C$ .

*Example 7.* Consider an instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  with  $\mathcal{F}_H = \{(b_1 \vee b_2)\}$ ,  $\mathcal{B}(\mathcal{F}) = \{b_1, b_2\}$  and  $w(b_1) = w(b_2) = 1$ . The clause  $C = (\neg b_1)$  is CPR wrt  $\mathcal{F}$ . In order to see this, let  $\tau = \{\neg b_1, b_2\}$ . Then  $\tau$  satisfies  $C$  (condition (i) of Definition 2). Furthermore,  $\tau$  satisfies  $\mathcal{F}_H$ , implying  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{\tau}$  (condition (ii)). Finally, we have that  $1 = \text{COST}(\mathcal{F}, \tau) = \text{COST}(\mathcal{F} \wedge C, \tau) \leq \text{COST}(\mathcal{F}, \neg C) = 1$  (condition (iii)). The assignment  $\delta = \{b_1, \neg b_2\}$  is an example of an optimal solution of  $\mathcal{F}$  that is not a solution of  $\mathcal{F} \wedge C$ .

## 4 Propagation Redundancy and MCSes

In this section, we analyze the effect of adding redundant clauses on the MCSes of MaxSAT instances. As the main result, we show that adding CSPR (and by extension CLPR) clauses to a MaxSAT instance  $\mathcal{F}$  preserves all MCSes while adding CPR clauses does not in general. Stated in terms of solutions, this means that adding CSPR clauses to  $\mathcal{F}$  preserves not only all optimal solutions, but all solutions  $\tau$  for which  $(\tau \cap \mathcal{B}(\mathcal{F})) \in \text{mcs}(\mathcal{F})$ , while adding CPR clauses only preserves at least one optimal solution.

**Effect of CLPR Clauses on MCSes.** MaxSAT-liftings of four specific SAT solving techniques (including bounded variable elimination and self-subsuming

resolution) were earlier proposed in [8]. Notably, the correctness of the liftings was shown individually for each of the techniques by arguing individually that applying one of the liftings does not change the set of MCSes of any MaxSAT instance. Towards a more generic understanding of optimal cost preserving MaxSAT preprocessing, in [10] the notion of solution resolution asymmetric tautologies (SRAT) was proposed as a MaxSAT-lifting of the concept of resolution asymmetric tautologies (RAT). In short, a clause  $C$  is a SRAT clause for a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  if there is a literal  $l \in C \setminus \mathcal{B}(\mathcal{F} \wedge C)$  such that  $\mathcal{F}_H \vdash_1 ((C \vee D) \setminus \{\neg l\})$  for every  $D \in \mathcal{F}_H$  for which  $\neg l \in D$ .

In analogy with RAT [29], SRAT was shown in [10] to allow for a general proof of correctness for natural MaxSAT-liftings of a wide range of SAT preprocessing techniques, covering among other the four techniques for which individual correctness proofs were provided in [8]. The generality follows essentially from the fact that the addition and removal of SRAT clauses preserves MCSes. The same observations apply to CLPR, as CLPR and SRAT are equivalent.

**Proposition 3.** *A clause  $C$  is CLPR wrt  $\mathcal{F}$  iff it is SRAT wrt  $\mathcal{F}$ .*

The proof of Proposition 3 follows directly from corresponding results in the context of SAT [23]. Informally speaking, a clause  $C$  is SRAT on a literal  $l$  iff it is RAT [29] on  $l$  and  $l \notin \mathcal{B}(\mathcal{F})$ . Similarly, a clause  $C$  is CLPR on a literal  $l$  iff it is LPR as defined in [23] on  $l$  and  $l \notin \mathcal{B}(\mathcal{F})$ . Proposition 3 together with previous results from [10] implies that the MCSes of MaxSAT instances are preserved under removing and adding CLPR clauses.

**Corollary 2.** *If  $C$  is CLPR wrt  $\mathcal{F}$ , then  $\text{mcs}(\mathcal{F}) = \text{mcs}(\mathcal{F} \wedge C)$ .*

**Effect of CPR Clauses on MCSes.** We turn our attention to the effect of CPR clauses on the MCSes of MaxSAT instances. Our analysis makes use of the previously-proposed MaxSAT-centric preprocessing rule known as *subsumed label elimination* (SLE) [11,33]<sup>5</sup>.

**Definition 3.** (*Subsumed Label Elimination* [11,33]) *Consider a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  and a blocking literal  $l \in \mathcal{B}(\mathcal{F})$  for which  $\neg l \notin \text{lit}(\mathcal{F}_H)$ . Assume that there is another blocking literal  $l_s \in \mathcal{B}(\mathcal{F})$  for which (1)  $\neg l_s \notin \text{lit}(\mathcal{F}_H)$ , (2)  $\{C \in \mathcal{F}_H \mid l \in C\} \subset \{C \in \mathcal{F}_H \mid l_s \in C\}$  and (3)  $w(l) \geq w(l_s)$ . The subsumed label elimination (SLE) rule allows adding  $(\neg l)$  to  $\mathcal{F}_H$ .*

A specific proof of correctness of SLE was given in [11]. The following proposition provides an alternative proof based on CPR.

**Proposition 4 (Proof of correctness for SLE).** *Let  $\mathcal{F}$  be a MaxSAT instance and assume that the blocking literals  $l, l_s \in \mathcal{B}(\mathcal{F})$  satisfy the three conditions of Definition 3. Then, the clause  $C = (\neg l)$  is CPR wrt  $\mathcal{F}$ .*

<sup>5</sup> Rephrased here using our notation.

*Proof.* We show that  $\tau = \{\neg l, l_s\}$  satisfies the three conditions of Definition 2. First  $\tau$  satisfies  $C$  (condition (i)). Conditions (1) and (2) of Definition 3 imply  $\mathcal{F}_H|_{\tau} \subset \mathcal{F}_H|_{\neg C}$  which in turn implies  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{\tau}$  (condition (ii)).

As for condition (iii), the requirement  $\text{COST}(\mathcal{F} \wedge C, \tau) = \text{COST}(\mathcal{F}, \tau)$  follows from  $\mathcal{B}(\mathcal{F} \wedge C) = \mathcal{B}(\mathcal{F})$ . Let  $\delta \supset \neg C$  be a complete assignment of  $\mathcal{F}_H$  for which  $\text{COST}(\mathcal{F}, \delta) = \text{COST}(\mathcal{F}, \neg C)$ . If  $\text{COST}(\mathcal{F}, \delta) = \infty$  then  $\text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \neg C)$  follows trivially. Otherwise  $\delta \setminus \neg C$  satisfies  $\mathcal{F}_H|_{\neg C}$  so by  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_{\tau}$  it satisfies  $\mathcal{F}_H|_{\tau}$  as well. Thus  $\delta^R = ((\delta \setminus \neg C) \setminus \{\neg l \mid l \in \tau\}) \cup \tau = (\delta \setminus \{l, \neg l, \neg l_s\}) \cup \{\neg l, l_s\}$  is an extension of  $\tau$  that satisfies  $\mathcal{F}_H$  and for which  $\text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \delta^R) \leq \text{COST}(\mathcal{F}, \delta)$  by condition (3) of Definition 3. Thereby  $\tau$  satisfies the conditions of Definition 2 so  $C$  is CPR wrt  $\mathcal{F}$ .  $\square$

*Example 8.* The blocking literals  $b_3, b_4 \in \mathcal{B}(\mathcal{F})$  of the instance  $\mathcal{F}$  detailed in Example 1 satisfy the conditions of Definition 3. By Proposition 4 the clause  $(\neg b_3)$  is CPR wrt  $\mathcal{F}$ .

In [11] it was shown that SLE does not preserve MCSes in general. By Corollary 2, this implies that SLE can not be viewed as the addition of CLPR clauses. Furthermore, by Proposition 4 we obtain the following.

**Corollary 3.** *There is a MaxSAT instance  $\mathcal{F}$  and a clause  $C$  that is CPR wrt  $\mathcal{F}$  for which  $\text{mcs}(\mathcal{F}) \neq \text{mcs}(\mathcal{F} \wedge C)$ .*

**Effect of CSPR Clauses on MCSes.** Having established that CLPR clauses preserve MCSes while CPR clauses do not, we complete the analysis by demonstrating that CSPR clauses preserve MCSes.

**Theorem 2.** *Let  $\mathcal{F}$  be a MaxSAT instance and  $C$  a CSPR clause of  $\mathcal{F}$ . Then  $\text{mcs}(\mathcal{F}) = \text{mcs}(\mathcal{F} \wedge C)$ .*

Theorem 2 follows from the following lemmas and propositions. In the following, let  $C$  be a clause that is CSPR wrt a MaxSAT instance  $\mathcal{F}$  on a set  $L \subset C \setminus \mathcal{B}(\mathcal{F} \wedge C)$ .

**Lemma 2.** *Let  $cs \subset \mathcal{B}(\mathcal{F})$ . If  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus cs} (\neg l)$  is satisfiable, then  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F} \wedge C) \setminus cs} (\neg l)$  is satisfiable.*

Lemma 2 helps in establishing one direction of Theorem 2.

**Proposition 5.**  $\text{mcs}(\mathcal{F}) \subset \text{mcs}(\mathcal{F} \wedge C)$ .

*Proof.* Let  $cs \in \text{mcs}(\mathcal{F})$ . Then  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus cs} (\neg l)$  is satisfiable, which by Lemma 2 implies that  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F} \wedge C) \setminus cs} (\neg l)$  is satisfiable.

To show that  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F} \wedge C) \setminus cs_s} (\neg l)$  is unsatisfiable for any  $cs_s \subsetneq cs \subset \mathcal{B}(\mathcal{F})$ , we note that any assignment satisfying  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F} \wedge C) \setminus cs_s} (\neg l)$  would also satisfy  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus cs_s} (\neg l)$ , contradicting  $cs \in \text{mcs}(\mathcal{F})$ .  $\square$

The following lemma is useful for showing inclusion in the other direction.

**Lemma 3.** *Let  $cs \in \text{mcs}(\mathcal{F} \wedge C)$ . Then  $cs \subset \mathcal{B}(\mathcal{F})$ .*

Lemma 3 allows for completing the proof of Theorem 2.

**Proposition 6.**  $\text{mcs}(\mathcal{F} \wedge C) \subset \text{mcs}(\mathcal{F})$ .

*Proof.* Let  $cs \in \text{mcs}(\mathcal{F} \wedge C)$ , which by Lemma 3 implies  $cs \subset \mathcal{B}(\mathcal{F})$ . Let  $\tau$  be a solution that satisfies  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F} \wedge C) \setminus cs} (\neg l)$ . Then  $\tau$  satisfies  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus cs} (\neg l)$ . For contradiction, assume that  $\mathcal{F}_H \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus cs_s} (\neg l)$  is satisfiable for some  $cs_s \subsetneq cs$ . Then by Lemma 2,  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F} \wedge C) \setminus cs_s} (\neg l)$  is satisfiable as well, contradicting  $cs \in \text{mcs}(\mathcal{F} \wedge C)$ . Thereby  $cs \in \text{mcs}(\mathcal{F})$ .  $\square$

Theorem 2 implies that SLE can not be viewed as the addition of CSPR clauses. In light of this, an interesting remark is that—in contrast to CPR clauses in general (recall Example 6)—the assignment  $\tau$  used in the proof of Proposition 4 can be used to convert any assignment that does not satisfy the CPR clause detailed in Definition 3 into one that does, without increasing its cost.

**Observation 4** *Let  $\mathcal{F}$  be a MaxSAT instance and assume that the blocking literals  $l, l_s \in \mathcal{B}(\mathcal{F})$  satisfy the three conditions of Definition 3. Let  $\tau = \{\neg l, l_s\}$  and consider any solution  $\delta \supset \neg C$  of  $\mathcal{F}$  that does not satisfy the CPR clause  $C = (\neg l)$ . Then  $\delta_\tau$  is a solution of  $\mathcal{F} \wedge C$  for which  $\text{COST}(\mathcal{F}, \delta_\tau) \leq \text{COST}(\mathcal{F}, \delta)$ .*

## 5 CPR-Based Preprocessing for MaxSAT

Mapping the theoretical observations into practical preprocessing, in this section we discuss through examples how CPR clauses can be used as a unified theoretical basis for capturing a wide variety of known MaxSAT reasoning rules, and how they could potentially help in the development of novel MaxSAT reasoning techniques.

Our first example is the so-called *hardening rule* [2, 8, 17, 26]. In terms of our notation, given a solution  $\tau$  to a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  and a blocking literal  $l \in \mathcal{B}(\mathcal{F})$  for which  $w(l) > \text{COST}(\mathcal{F}, \tau)$ , the hardening rule allows adding the clause  $C = (\neg l)$  to  $\mathcal{F}_H$ .

The correctness of the hardening rule can be established with CPR clauses. More specifically, as  $\text{COST}(\mathcal{F}, \tau) < w(l)$  it follows that  $\tau(C) = 1$  (condition (i) of Definition 2). Since  $\tau$  satisfies  $\mathcal{F}$ , we have that  $\mathcal{F}_H|_\tau = \top$  so  $\mathcal{F}_H|_{\neg C} \vdash_1 \mathcal{F}_H|_\tau$  (condition (ii)). Finally, as  $\text{COST}(\mathcal{F}, \delta) \geq w(l) > \text{COST}(\mathcal{F}, \tau)$  holds for all  $\delta \supset \neg C$  it follows that  $\text{COST}(\mathcal{F}, \neg C) > \text{COST}(\mathcal{F}, \tau) = \text{COST}(\mathcal{F} \wedge C, \tau)$ . As such,  $(\neg l)$  is CPR clause wrt  $\mathcal{F}$ . If fact, instead of assuming  $w(l) > \text{COST}(\mathcal{F}, \tau)$  it suffices to assume  $w(l) \geq \text{COST}(\mathcal{F}, \tau)$  and  $\tau(l) = 0$ .

The hardening rule can not be viewed as the addition of CSPR or CLPR clauses because it does not in general preserve MCSes.

*Example 9.* Consider the MaxSAT instance  $\mathcal{F}$  from Example 1 and a solution  $\tau = \{b_1, b_2, b_4, \neg b_3, \neg z, x, y\}$ . Since  $\text{COST}(\mathcal{F}, \tau) = 3 < 8 = w(b_3)$ , the clause  $(\neg b_3)$  is CPR. However,  $\text{mcs}(\mathcal{F}) \neq \text{mcs}(\mathcal{F} \wedge C)$  since the set  $\{b_2, b_3\} \in \text{mcs}(\mathcal{F})$  is not an MCS of  $\mathcal{F} \wedge C$  as  $(\mathcal{F}_H \wedge C) \wedge \bigwedge_{l \in \mathcal{B}(\mathcal{F}) \setminus cs} (\neg l) = (\mathcal{F}_H \wedge (\neg b_3)) \wedge (\neg b_1) \wedge (\neg b_4)$  is not satisfiable.

Viewing the hardening rule through the lens of CPR clauses demonstrates novel aspects of the MaxSAT-liftings of propagation redundancy. In particular, instantiated in the context of SAT, an argument similar to the one we made for hardening shows that given a CNF formula  $F$ , an assignment  $\tau$  satisfying  $F$ , and a literal  $l$  for which  $\tau(l) = 0$ , the clause  $(\neg l)$  is redundant (wrt equisatisfiability). While formally correct, such a rule is not very useful for SAT solving. In contrast, in the context of MaxSAT the hardening rule is employed in various modern MaxSAT solvers and leads to non-trivial performance-improvements [4, 5].

As another example of capturing MaxSAT-centric reasoning with CPR, consider the so-called TrimMaxSAT rule [39]. Given a MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  and a literal  $l \in \mathcal{B}(\mathcal{F})$  for which  $\tau(l) = 1$  for all solutions of  $\mathcal{F}$ , the TrimMaxSAT rule allows adding the clause  $C = (l)$  to  $\mathcal{F}_H$ . In this case the assumptions imply that all solutions of  $\mathcal{F}$  also satisfy  $C$ , i.e., that  $\mathcal{F}_H|_{\neg C}$  is unsatisfiable. As such, any assignment  $\tau$  that satisfies  $C$  and  $\mathcal{F}_H$  will also satisfy the three conditions of Definition 2 which demonstrates that  $C$  is CPR. It is, however, not CSPP since the only literal in  $C$  is blocking.

As a third example of capturing (new) reasoning techniques with CPR, consider an extension of the central variable elimination rule that allows (to some extent) for eliminating blocking literals.

**Definition 4.** Consider a MaxSAT instance  $\mathcal{F}$  and a blocking literal  $l \in \mathcal{B}(\mathcal{F})$ . Let  $\text{BBVE}(\mathcal{F})$  be the instance obtained by (i) adding the clause  $C \vee D$  to  $\mathcal{F}$  for every pair  $(C \vee l), (D \vee \neg l) \in \mathcal{F}_H$  and (ii) removing all clauses  $(D \vee \neg l) \in \mathcal{F}_H$ . Then  $\text{COST}(\mathcal{F}) = \text{COST}(\text{BBVE}(\mathcal{F}))$  and  $\text{mcs}(\mathcal{F}) = \text{mcs}(\text{BBVE}(\mathcal{F}))$ .

**On the Limitations of CPR.** Finally, we note that while CPR clauses significantly generalize existing theory on reasoning and preprocessing rules for MaxSAT, there are known reasoning techniques that can not (at least straightforwardly) be viewed through the lens of propagation redundancy. For a concrete example, consider the so-called intrinsic atmost1 technique [26].

**Definition 5.** Consider a MaxSAT instance  $\mathcal{F}$  and a set  $L \subset \mathcal{B}(\mathcal{F})$  of blocking literals. Assume that (i)  $|\tau \cap \{\neg l \mid l \in L\}| \leq 1$  holds for any solution  $\tau$  of  $\mathcal{F}$  and (ii)  $w(l) = 1$  for each  $l \in L$ . Now form the instance  $\text{AT-MOST-ONE}(\mathcal{F}, L)$  by (i) removing each literal  $l \in L$  from  $\mathcal{B}(\mathcal{F})$ , and (ii) adding the clause  $\{(\neg l) \mid l \in L\} \cup \{l_L\}$  to  $\mathcal{F}$ , where  $l_L$  is a fresh blocking literal with  $w(l_L) = 1$ .

It has been established that any optimal solution of  $\text{AT-MOST-ONE}(\mathcal{F}, L)$  is an optimal solution of  $\mathcal{F}$  [26]. However, as the next example demonstrates, the preservation of optimal solutions is in general not due to the clauses added being redundant, as applying the technique can affect optimal cost.

*Example 10.* Consider the MaxSAT instance  $\mathcal{F} = (\mathcal{F}_H, \mathcal{B}(\mathcal{F}), w)$  with  $\mathcal{F}_H = \{(l_i) \mid i = 1 \dots n\}$ ,  $\mathcal{B}(\mathcal{F}) = \{l_1 \dots l_n\}$  and  $w(l) = 1$  for all  $l \in \mathcal{B}(\mathcal{F})$ . Then  $|\tau \cap \neg \mathcal{B}(\mathcal{F})| = 0 \leq 1$  holds for all solutions  $\tau$  of  $\mathcal{F}$  so the intrinsic-at-most-one technique can be used to obtain the instance  $\mathcal{F}^2 = \text{AT-MOST-ONE}(\mathcal{F}, \mathcal{B}(\mathcal{F})) = (\mathcal{F}_H^2, \mathcal{B}(\mathcal{F}^2), w^2)$  with  $\mathcal{F}_H^2 = \mathcal{F}_H \cup \{(\neg l_1 \vee \dots \vee \neg l_n \vee l_L)\}$ ,  $\mathcal{B}(\mathcal{F}^2) = \{l_L\}$  and

$w^2(l_L) = 1$ . Now  $\delta = \{l \mid l \in \mathcal{B}(\mathcal{F})\} \cup \{l_L\}$  is an optimal solution to both  $\mathcal{F}^2$  and  $\mathcal{F}$  for which  $1 = \text{COST}(\mathcal{F}^2, \delta) < \text{COST}(\mathcal{F}, \delta) = n$ .

Example 10 implies that the intrinsic atleast1 technique can not be viewed as the addition or removal of redundant clauses. Generalizing CPR to cover weight changes could lead to further insights especially due to potential connections with core-guided MaxSAT solving [1, 36–38].

## 6 MaxPre 2: More General Preprocessing in Practice

Connecting to practice, we extended the MaxSAT preprocessor MaxPre [33] version 1 with support for techniques captured by propagation redundancy. The resulting MaxPre version 2, as outlined in the following, hence includes techniques which have previously only been implemented in specific solver implementations rather than in general-purpose MaxSAT preprocessors.

First, let us mention that the earlier MaxPre [33] version 1 assumes that any blocking literals only appear in a single polarity among the hard clauses. Removing this assumption—supported by theory developed in Sects. 3–4—decreases the number of auxiliary variables that need to be introduced when a MaxSAT instance is rewritten to only include unit soft clauses. For example, consider a MaxSAT instance  $\mathcal{F}$  with  $\mathcal{F}_H = \{(\neg x \vee y), (\neg y \vee x)\}$  and  $\mathcal{F}_S = \{(x), (\neg y)\}$ . For preprocessing the instance, MaxPre 1 extends both soft clauses with a new, auxiliary variable and runs preprocessing on the instance  $\mathcal{F} = \{(\neg x \vee y), (\neg y \vee x), (x \vee b_1), (\neg y \vee b_2)\}$  with  $\mathcal{B}(\mathcal{F}) = \{b_1, b_2\}$ . In contrast, MaxPre 2 detects that the clauses in  $\mathcal{F}_S$  are unit and reuses them as blocking literals, invoking preprocessing on  $\mathcal{F} = \{(\neg x \vee y), (\neg y \vee x)\}$  with  $\mathcal{B}(\mathcal{F}) = \{\neg x, y\}$ .

In addition to the techniques already implemented in MaxPre 1, MaxPre 2 includes the following additional techniques: hardening [2], a variant TrimMaxSAT [39] that works on all literals of a MaxSAT instance, the intrinsic atleast1 technique [26] and a MaxSAT-lifting of failed literal elimination [12]. In short, failed literal elimination adds the clause  $(\neg l)$  to the hard clauses  $\mathcal{F}_H$  of an instance in case unit-propagation derives a conflict in  $\mathcal{F}_H \wedge \{(l)\}$ . Additionally, the implementation of failed literal elimination attempts to identify implied equivalences between literals that can lead to further simplification.

For computing the solutions required by TrimMaxSAT and detecting the cardinality constraints required by intrinsic-at-most-one constraints, MaxPre 2 uses the Glucose 3.0 SAT-solver [3]. For computing solutions required by hardening, MaxPre 2 additionally uses the SatLike incomplete MaxSAT solver [34] within preprocessing. MaxPre 2 is available in open source at <https://bitbucket.org/coreo-group/maxpre2/>.

We emphasize that, while the additional techniques implemented by MaxPre 2 have been previously implemented as heuristics in specific solver implementations, MaxPre 2 is—to the best of our understanding—the first stand-alone implementation supporting techniques whose correctness cannot be established with previously-proposed MaxSAT redundancy notions (i.e., SRAT). The goal

of our empirical evaluation presented in the next section is to demonstrate the potential of viewing expressive reasoning techniques not only as solver heuristics, but as a separate step in the MaxSAT solving process whose correctness can be established via propagation redundancy.

## 7 Empirical Evaluation

We report on results from an experimental evaluation of the potential of incorporating more general reasoning in MaxSAT preprocessing. In particular, we evaluated both complete solvers (geared towards finding provably-optimal solutions) and incomplete solvers (geared towards finding relatively good solutions fast) on standard heterogeneous benchmarks from recent MaxSAT Evaluations. All experiments were run on 2.60-GHz Intel Xeon E5-2670 8-core machines with 64 GB memory and CentOS 7. All reported runtimes include the time used in preprocessing (when applicable).

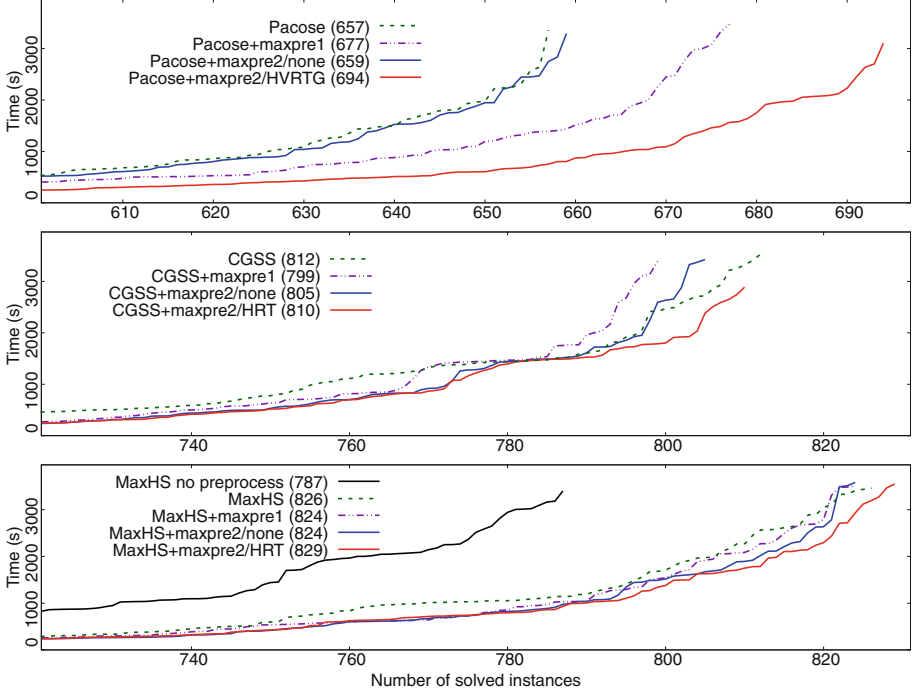
### 7.1 Impact of Preprocessing on Complete Solvers

We start by considering recent representative complete solvers covering three central MaxSAT solving paradigms: the core-guided solver CGSS [27] (as a recent improvement to the successful RC2 solver [26]), and the MaxSAT Evaluation 2021 versions of the implicit hitting set based solver MaxHS [17] and the solution-improving solver Pacose [40]. For each solver  $S$  we consider the following variants.

- $S$ :  $S$  in its default configuration.
- $S$  **no preprocess**:  $S$  with the solver’s own internal preprocessing turned off (when applicable).
- $S$ +**maxpre1**:  $S$  after applying MaxPre 1 using its default configuration.
- $S$ +**maxpre2/none**:  $S$  after applying MaxPre 2 using the default configuration of MaxPre 1.
- $S$  + **maxpre2/⟨TECH⟩**:  $S$  after applying MaxPre 2 using the standard configuration of MaxPre 1 and additional techniques integrated into MaxPre 2 (as detailed in Section 6) as specified by ⟨TECH⟩.

More precisely, ⟨TECH⟩ specifies which of the techniques HTVGR are applied: H for hardening, T and V for TrimMaxSAT on blocking and non-blocking literals, respectively, G for intrinsic-at-most-one-constraints and R for failed literal elimination. It should be noted that an exhaustive evaluation of all subsets and application orders of these techniques is infeasible in practice. Based on preliminary experiments, we observed that the following choices were promising: HRT for CGSS and MaxHS, and HTVGR for Pacose; we report results using these individual configurations.

As benchmarks, we used the combined set of weighted instances from the complete tracks of MaxSAT Evaluation 2020 and 2021. After removing duplicates, this gave a total of 1117 instances. We enforced a per-instance time limit of



**Fig. 1.** Impact of preprocessing on complete solvers. For each solver, the number of instances solved within a 60-min per-instance time limit in parentheses.

60 minutes and memory limit of 32 GB. Furthermore, we enforced a per-instance 120-second time limit on preprocessing.

An overview of the results is shown in Fig. 1, illustrating for each solver the number of instances solved (x-axis) under different per-instance time limits (y-axis). We observe that for both CGSS and MaxHS,  $S+\text{maxpre1}$  and  $S+\text{maxpre2/none}$  leads to less instances solved compared to  $S$ . In contrast,  $S+\text{maxpre2/HRT}$ , i.e., incorporating the stronger reasoning techniques of MaxPre 2, performs best of all preprocessing variants and improves on MaxHS also in terms of the number of instances solved. For Pacose, we observe that both  $\text{Pacose}+\text{maxpre1}$  and  $\text{Pacose}+\text{maxpre2/new}$  (without the stronger reasoning techniques) already improve the performance of Pacose, leading to more instances solved. Incorporating the stronger reasoning rules further significantly improves performance, with  $\text{Pacose}+\text{maxpre2/HVRTG}$  performing the best among all of the Pacose variants.

## 7.2 Impact of Preprocessing on Incomplete MaxSAT Solving

As a representative incomplete MaxSAT solver we consider the MaxSAT Evaluation 2021 version of Loandra [9], as the best-performing solver in the incomplete



**Table 1.** Impact of preprocessing on the incomplete solver Loandra. The wins are organized column-wise, the cell on row  $X$  column  $Y$  contains the total number of instances that the solver on column  $Y$  wins over the solver on row  $X$ .

#Wins	base (maxpre1)	no-prepro	maxpre2/ none	maxpre2/ VG
base (maxpre1)	—	154	135	152
no-prepro	208	—	216	218
maxpre2/none	105	143	—	77
maxpre2/VG	110	140	80	—
<b>Score (avg):</b>	0.852	0.840	0.863	0.870

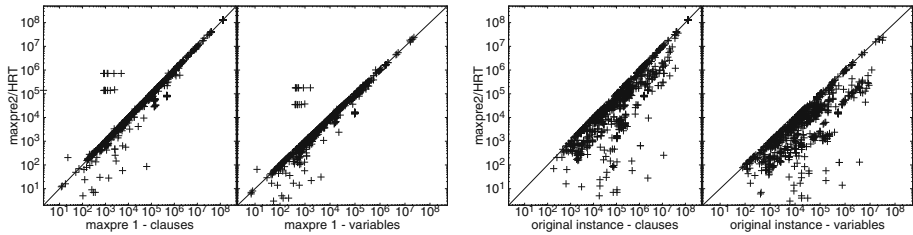
track of MaxSAT Evaluation under a 300s per-instance time limit on weighted instances. Loandra combines core-guided and solution-improving search towards finding good solutions fast. We consider the following variants of Loandra.

- **base (maxpre1)**: Loandra in its default configuration which makes use of MaxPre 1.
- **no-prepro**: Loandra with its internal preprocessing turned off.
- **maxpre2/none**: **base** with its internal preprocessor changed from MaxPre 1 to MaxPre 2 using the default configuration of MaxPre 1.
- **maxpre2/VG**: **maxpre2** incorporating the additional intrinsic-at-most-one constraints technique and the extension of TrimMaxSAT to non-blocking literals (cf. Sect. 6), found promising in preliminary experimentation.

As benchmarks, we used the combined set of weighted instances from the incomplete tracks of MaxSAT Evaluation 2020 and 2021. After removing duplicates, this gave a total of 451 instances. When reporting results, we consider for each instance and solver the cost of the best solution found by the solver within 300 s (including time spent preprocessing and solution reconstruction).

We compare the relative runtime performance of the solver variants using two metrics: *#wins* and the average *incomplete score*. Assume that  $\tau_x$  and  $\tau_y$  are the lowest-cost solutions computed by two solvers  $X$  and  $Y$  on a MaxSAT instance  $\mathcal{F}$  and that  $\text{best-cost}(\mathcal{F})$  is the lowest cost of a solution of  $\mathcal{F}$  found either in our evaluation or in the MaxSAT Evaluations. Then  $X$  wins over  $Y$  if  $\text{COST}(\mathcal{F}, \tau_x) < \text{COST}(\mathcal{F}, \tau_y)$ . The incomplete score,  $\text{score}(\mathcal{F}, X)$ , obtained by solver  $X$  on  $\mathcal{F}$  is the ratio between the cost of the solution found by  $X$  and  $\text{best-cost}(\mathcal{F})$ , i.e.,  $\text{score}(\mathcal{F}, X) = (\text{best-cost}(\mathcal{F}) + 1) / (\text{COST}(\mathcal{F}, \tau_x) + 1)$ . The score of  $X$  on  $\mathcal{F}$  is 0 if  $X$  is unable to find any solutions within 300 s.

An overview of the results is shown in Table 1. The upper part of the table shows a pairwise comparison on the number of wins over all benchmarks. The wins are organized column-wise, i.e., the cell on row  $X$  column  $Y$  contains the total number of instances that the solver on column  $Y$  wins over the solver on row  $X$ . The last row contains the average score obtained by each solver over all instances. We observe that any form of preprocessing improves the performance of Loandra, as witnessed by the fact that **no-prepro** is clearly the worst-performing variant. The variants that make use of MaxPre 2 outperform the



**Fig. 2.** Impact of preprocessing on instance size.

baseline under both metrics; both `maxpre2 no new` and `maxpre2-w:VG` obtain a higher average score and win on more instances over `base`. The comparison between `maxpre2/none` and `maxpre2/VG` is not as clear. On one hand, the score obtained by `maxpre2/VG` is higher. On the other hand, `maxpre2/none` wins on 80 instances over `maxpre2/VG` and loses on 77. This suggests that the quality of solutions computed by `maxpre2/VG` is on average higher, and that on the instances on which `maxpre2/none` wins the difference is smaller.

### 7.3 Impact of Preprocessing on Instance Sizes

In addition to improved solver runtimes, we note that MaxPre 2 has a positive effect on the size of instances (both in terms of the number of variables and clauses remaining) when compared to preprocessing with MaxPre 1; see Fig. 2 for a comparison, with `maxpre2/HRT` compared to `maxpre1` (left) and to original instance sizes (right).

## 8 Conclusions

We studied liftings of variants of propagation redundancy from SAT in the context of maximum satisfiability where—more fine-grained than in SAT—of interest are reasoning techniques that preserve optimal cost. We showed that CPR, the strongest MaxSAT-lifting, allows for changing minimal corrections sets in MaxSAT in a controlled way, thereby succinctly expressing MaxSAT reasoning techniques very generally. We also provided a practical MaxSAT preprocessor extended with techniques captured by CPR and showed empirically that extended preprocessing has a positive overall impact on a range of MaxSAT solvers. Interesting future work includes the development of new CPR-based preprocessing rules for MaxSAT capable of significantly affecting the MaxSAT solving pipeline both in theory and practice, as well as developing an understanding of the relationship between redundancy notions and the transformations performed by MaxSAT solving algorithms.

## References

1. Ansótegui, C., Bonet, M., Levy, J.: SAT-based MaxSAT algorithms. *Artif. Intell.* **196**, 77–105 (2013)

2. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-based weighted MaxSAT solvers. In: Milano, M. (ed.) CP 2012. LNCS, pp. 86–101. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-33558-7\\_9](https://doi.org/10.1007/978-3-642-33558-7_9)
3. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the IJCAI, pp. 399–404 (2009)
4. Bacchus, F., Berg, J., Järvisalo, M., Martins, R. (eds.): MaxSAT Evaluation 2020: Solver and Benchmark Descriptions, Department of Computer Science Report Series B, vol. B-2020-2. Department of Computer Science, University of Helsinki (2020)
5. Bacchus, F., Järvisalo, M., Martins, R. (eds.): MaxSAT Evaluation 2019: Solver and Benchmark Descriptions, Department of Computer Science Report Series B, vol. B-2019-2. Department of Computer Science, University of Helsinki (2019)
6. Bacchus, F., Järvisalo, M., Martins, R.: Maximum satisfiability (chap. 24). In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, pp. 929–991. IOS Press (2021)
7. Baek, S., Carneiro, M., Heule, M.J.H.: A flexible proof format for sat solver-elaborator communication. In: TACAS 2021. LNCS, vol. 12651, pp. 59–75. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-72016-2\\_4](https://doi.org/10.1007/978-3-030-72016-2_4)
8. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for MaxSAT. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 96–111. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-45221-5\\_7](https://doi.org/10.1007/978-3-642-45221-5_7)
9. Berg, J., Demirović, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 39–56. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-19212-9\\_3](https://doi.org/10.1007/978-3-030-19212-9_3)
10. Berg, J., Järvisalo, M.: Unifying reasoning and core-guided search for maximum satisfiability. In: Calimeri, F., Leone, N., Manna, M. (eds.) JELIA 2019. LNCS (LNAI), vol. 11468, pp. 287–303. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-19570-0\\_19](https://doi.org/10.1007/978-3-030-19570-0_19)
11. Berg, J., Saikko, P., Järvisalo, M.: Subsumed label elimination for maximum satisfiability. In: Proceedings of the ECAI. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 630–638. IOS Press (2016)
12. Bhalla, A., Lynce, I., de Sousa, J.T., Marques-Silva, J.: Heuristic-based backtracking for propositional satisfiability. In: Pires, F.M., Abreu, S. (eds.) EPIA 2003. LNCS (LNAI), vol. 2902, pp. 116–130. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-24580-3\\_19](https://doi.org/10.1007/978-3-540-24580-3_19)
13. Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability (Second Edition): Volume 336 Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, The Netherlands (2021)
14. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving (chap. 9). In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, pp. 391–435. IOS Press (2021)
15. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 220–236. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63046-5\\_14](https://doi.org/10.1007/978-3-319-63046-5_14)
16. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 118–135. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54577-5\\_7](https://doi.org/10.1007/978-3-662-54577-5_7)

17. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MAXSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39071-5\\_13](https://doi.org/10.1007/978-3-642-39071-5_13)
18. Heule, M., Hunt, W., Kaufmann, M., Wetzler, N.: Efficient, verified checking of propositional proofs. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP 2017. LNCS, vol. 10499, pp. 269–284. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66107-0\\_18](https://doi.org/10.1007/978-3-319-66107-0_18)
19. Heule, M., Hunt, W.A., Jr., Wetzler, N.: Bridging the gap between easy generation and efficient verification of unsatisfiability proofs. *Softw. Test. Verif. Reliab.* **24**(8), 593–607 (2014)
20. Heule, M., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. *J. Artif. Intell. Res.* **53**, 127–168 (2015)
21. Heule, M., Kiesl, B.: The potential of interference-based proof systems. In: Proceedings of the ARCADE@CADE. EPiC Series in Computing, vol. 51, pp. 51–54. EasyChair (2017)
22. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 130–147. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63046-5\\_9](https://doi.org/10.1007/978-3-319-63046-5_9)
23. Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension-free proof systems. *J. Autom. Reason.* **64**(3), 533–554 (2020)
24. Heule, M.J.H., Kiesl, B., Seidl, M., Biere, A.: PRuning through satisfaction. In: HVC 2017. LNCS, vol. 10629, pp. 179–194. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70389-3\\_12](https://doi.org/10.1007/978-3-319-70389-3_12)
25. Hou, A.: A theory of measurement in diagnosis from first principles. *Artif. Intell.* **65**(2), 281–328 (1994)
26. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an efficient MaxSAT solver. *J. Satisf. Boolean Model. Comput.* **11**(1), 53–64 (2019)
27. Ihalainen, H., Berg, J., Järvisalo, M.: Refined core relaxation for core-guided MaxSAT solving. In: Proceedings of the CP. LIPIcs, vol. 210, pp. 28:1–28:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
28. Järvisalo, M., Biere, A., Heule, M.: Simulating circuit-level simplifications on CNF. *J. Autom. Reason.* **49**(4), 583–619 (2012)
29. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 355–370. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31365-3\\_28](https://doi.org/10.1007/978-3-642-31365-3_28)
30. Kiesl, B., Rebola-Pardo, A., Heule, M.J.H., Biere, A.: Simulating strong practical proof systems with extended resolution. *J. Autom. Reason.* **64**(7), 1247–1267 (2020)
31. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 45–61. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-40229-1\\_5](https://doi.org/10.1007/978-3-319-40229-1_5)
32. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Local redundancy in SAT: generalizations of blocked clauses. *Log. Methods Comput. Sci.* **14**(4) (2018)
33. Korhonen, T., Berg, J., Saikko, P., Järvisalo, M.: MaxPre: an extended MaxSAT preprocessor. In: Gaspers, S., Walsh, T. (eds.) SAT 2017. LNCS, vol. 10491, pp. 449–456. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66263-3\\_28](https://doi.org/10.1007/978-3-319-66263-3_28)
34. Lei, Z., Cai, S.: Solving (weighted) partial MaxSAT by dynamic local search for SAT. In: Proceedings of the IJCAI, pp. 1346–1352 (2018). [ijcai.org](https://doi.org/10.1007/978-3-319-66263-3_28)
35. Li, C., Manyà, F.: MaxSAT, hard and soft constraints. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, pp. 613–631. IOS Press (2009)

36. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O’Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10428-7\\_41](https://doi.org/10.1007/978-3-319-10428-7_41)
37. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: a survey and assessment. *Constraints* **18**(4), 478–534 (2013)
38. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: *Proceedings of the AAAI*, pp. 2717–2723. AAAI Press (2014)
39. Paxian, T., Raiola, P., Becker, B.: On preprocessing for weighted MaxSAT. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) VMCAI 2021. LNCS, vol. 12597, pp. 556–577. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67067-2\\_25](https://doi.org/10.1007/978-3-030-67067-2_25)
40. Paxian, T., Reimer, S., Becker, B.: Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 37–53. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-94144-8\\_3](https://doi.org/10.1007/978-3-319-94144-8_3)
41. Rebola-Pardo, A., Cruz-Filipe, L.: Complete and efficient DRAT proof checking. In: *Proceedings of the FMCAD*, pp. 1–9. IEEE (2018)
42. Yolcu, E., Wu, X., Heule, M.J.H.: Mycielski graphs and PR proofs. In: Pulina, L., Seidl, M. (eds.) SAT 2020. LNCS, vol. 12178, pp. 201–217. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51825-7\\_15](https://doi.org/10.1007/978-3-030-51825-7_15)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description)

Gereon Kremer<sup>1</sup>, Andrew Reynolds<sup>2</sup>, Clark Barrett<sup>1</sup>,  
and Cesare Tinelli<sup>2</sup>

<sup>1</sup> Stanford University, Stanford, USA

<sup>2</sup> The University of Iowa, Iowa City, USA

[andrew.j.reynolds@gmail.com](mailto:andrew.j.reynolds@gmail.com)

**Abstract.** The `cvc5` SMT solver solves quantifier-free nonlinear real arithmetic problems by combining the cylindrical algebraic coverings method with incremental linearization in an abstraction-refinement loop. The result is a complete algebraic decision procedure that leverages efficient heuristics for refining candidate models. Furthermore, it can be used with quantifiers, integer variables, and in combination with other theories. We describe the overall framework, individual solving techniques, and a number of implementation details. We demonstrate its effectiveness with an evaluation on the SMT-LIB benchmarks.

**Keywords:** Satisfiability modulo theories · Nonlinear real arithmetic · Abstraction refinement · Cylindrical algebraic coverings

## 1 Introduction

SMT solvers are used as back-end engines for a wide variety of academic and industrial applications [2, 19, 20]. Efficient reasoning in the theory of real arithmetic is crucial for many such applications [5, 8]. While modern SMT solvers have been shown to be quite effective at reasoning about *linear* real arithmetic problems [21, 43], *nonlinear* problems are typically much more difficult. This is not surprising, given that the worst-case complexity for deciding the satisfiability of nonlinear real arithmetic formulas is doubly-exponential in the number of variables in the formula [15]. Nevertheless, a variety of techniques have been proposed and implemented, each attempting to target a class of formulas for which reasonable performance can be observed in practice.

*Related Work.* All complete decision procedures for nonlinear real arithmetic (or the *theory of the reals*) originate in computer algebra, the most prominent being cylindrical algebraic decomposition (CAD) [11]. While alternatives exist [6, 25, 41], they have not seen much use [27], and CAD-based methods are the only sound and complete methods in practical use today. CAD-based methods used

in modern SMT solvers include incremental CAD implementations [34, 36] and cylindrical algebraic coverings [3], both of which are integrated in the traditional CDCL(T) framework for SMT [40].

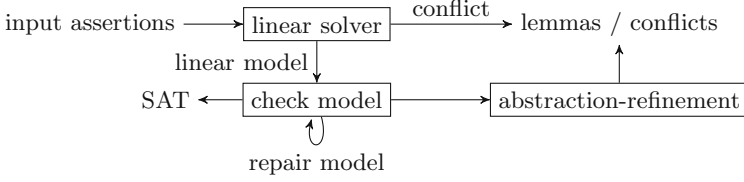
In contrast, the NLSAT [30] calculus and the generalized MCSAT [28, 39] framework provide for a much tighter integration of a conflict-driven CAD-based theory solver into a theory-aware core solver. This has been the dominant approach over the last decade due to its strong performance in practice. However, it has the significant disadvantage of being difficult to integrate with CDCL(T)-based frameworks for theory combination.

A number of *incomplete* techniques are also used by various SMT solvers: incremental linearization [9] gradually refines an abstraction of the nonlinear formula obtained via a naive linearization by refuting spurious models of the abstraction; interval constraint propagation [24, 36, 45] employs interval arithmetic to narrow down the search space; subtropical satisfiability [22] provides sufficient linear conditions for nonlinear solutions in the exponent space of the polynomials; and virtual substitution [12, 31, 46] makes use of parametric solution formulas for polynomials of bounded degree. Though all of these techniques have limitations, each of them is useful for certain subclasses of nonlinear real arithmetic or in combination with other techniques.

*Contributions.* We present an integration of cylindrical algebraic coverings and incremental linearization, implemented in the `cvc5` SMT solver. Crucial to the success of the integration is an abstraction-refinement loop used to combine the two techniques cooperatively. The solution is effective in practice, as witnessed by the fact that `cvc5` won the nonlinear real arithmetic category of SMT-COMP 2021 [44], the first time a non-MCSAT-based technique has won since 2013. Our integrated technique also has the advantage of being very flexible: in particular, it fits into the regular CDCL(T) schema for theory solvers and theory combination, it supports (mixed) integer problems, and it can be easily extended using further subsolvers that support additional arithmetic operators beyond the scope of traditional algebraic routines (e.g., transcendental functions).

## 2 Nonlinear Solving Techniques

The nonlinear arithmetic solver implemented in `cvc5` generally follows the abstraction-refinement framework introduced by Cimatti et al. [9] and depicted in Fig. 1. The input assertions are first checked by the linear arithmetic solver, where they are linearized implicitly by treating every application of multiplication as if it were an arithmetic variable. For example, given input assertions  $x \cdot y > 0 \wedge x > 1 \wedge y < 0$ , the linear solver treats the expression  $x \cdot y$  as a variable. It may then find the (spurious) model:  $x \mapsto 2$ ,  $y \mapsto -1$ , and  $x \cdot y \mapsto 1$ . We call the candidate model returned by the linear arithmetic solver, where applications of multiplication are treated as variables, a *linear model*. If a linear model does not exist, i.e., the input is unsatisfiable according to the linear solver, the linear solver generates a conflict that is immediately returned to the CDCL(T) engine.



**Fig. 1.** Structural overview of the nonlinear solver

When a linear model does exist, we check whether it already satisfies the input assertions or try to *repair* it to do so. We only apply a few very simple heuristics for repairs such as updating the value for  $z$  in the presence of a constraint like  $z = x \cdot y$  based on the values of  $x$  and  $y$ .

If the model can not be repaired, we refine the abstraction for the linear solver [9]. This step constructs lemmas, or conflicts, based on the input assertions and the linear model, to advance the solving process by blocking either the current linear model or the current Boolean model, that is, the propositional assignment generated by the SMT solver’s SAT engine. The Boolean model is usually eliminated only by the coverings approach, while the incremental linearization technique generates lemmas with new literals that target the linear model, e.g., the lemma  $x > 0 \wedge y < 0 \Rightarrow x \cdot y < 0$  in the example above. We next describe our implementation of cylindrical algebraic coverings and incremental linearization, and how they are combined in `cvc5`.

## 2.1 Cylindrical Algebraic Coverings

Cylindrical algebraic coverings is a technique recently proposed by Ábrahám et al. [3] and is heavily inspired by CAD. While the way the computation proceeds is very different from traditional CAD, and instead somewhat similar to NLSAT [30], their mathematical underpinnings are essentially identical. The cylindrical algebraic coverings subsolver in `cvc5` closely follows the presentation in [3]. Below, we discuss some differences and extensions. For this discussion, we must refer the reader to [3] for the relevant background material because of space constraints. We note that `cvc5` relies on the libpoly library [29] to provide most of the computational infrastructure for algebraic reasoning.

*Square-Free Basis.* As with most CAD projection schemas, the set of projection polynomials needs to be a square-free basis when computing the characterization for an interval in [3, Algorithm 4]. However, the resultants computed in this algorithm combine polynomials from different sets, which are not necessarily coprime. The remedy is to either make these sets of polynomials pairwise square-free or to fully factor all projection polynomials. We adopt the former approach.

*Starting Model.* Although the linear model may not satisfy the nonlinear constraints, we may expect it to be in the vicinity of a proper model. We thus



optionally use the linear model as an *initial assignment* for the cylindrical algebraic coverings algorithm in one of two ways: either using it initially in the search and discarding it as soon as it conflicts; or using it whenever possible, even if it leads to a conflict in another branch of the search. Unfortunately, neither technique has any discernible impact in our experiments.

*Interval Pruning.* As already noted in [3], a covering may contain two kinds of redundant intervals: intervals fully contained in another interval, or intervals contained in the union of other intervals. Removing the former kind of redundancies is not only clearly beneficial, but also required for how the characterizations are computed. It is not clear, however, if it is worthwhile to remove redundancies of the second kind because, while it can simplify the characterization locally, it may also make the resulting interval smaller, slowing down the overall solving process. Note that there may not be a unique redundant interval: e.g., if multiple intervals overlap, it may be possible to remove one of two intervals, but not both of them. We have implemented a simple heuristic to detect redundancies of the second kind, always removing the smallest interval with respect to the interval ordering given in [3]. Even if these redundancies occur in about 7.5% of all **QF\_NRA** benchmarks, using this technique has only a very limited impact. It may be that for certain kinds of benchmarks, underrepresented in SMT-LIB, the technique is valuable. Or it may be that some variation of the technique is more broadly helpful. These are interesting directions for future work.

*Lifting and Coefficient Selection with Lazard.* The original cylindrical algebraic coverings technique is based on McCallum’s projection operator [37], which is particularly well-studied, but also (refutationally) unsound: polynomial nullification may occur when computing the real roots, possibly leading to the loss of real roots and thus solution candidates. One then needs to check for these cases and fall back to a more conservative, albeit more costly, projection schema such as those due to Collins [11] or Hong [26].

Lazard’s projection schema [35], which has been proven correct only recently [38], provides very small projection sets and is both sound and complete. This comes at the price of a different mathematical background and a modified lifting procedure, which corresponds to a modified procedure for real root isolation. Although the local projections employed in cylindrical algebraic coverings have not been formally verified for Lazard’s projection schema yet, we expect no significant issues there. Adopting it seems to be a logical improvement, as already mentioned in [3]. The modified real root isolation procedure is a significant hurdle in practice, as it requires additional nontrivial algorithms [32, Section 5.3.2]. We implemented it using CoCoALib [1] in **cvc5** [33], achieving soundness without any discernible negative performance impact.

Using Lazard’s projection schema, for all its benefits, may seem questionable for the following reasons: (i) the unsoundness of McCallum’s projection operator is virtually never witnessed in practice [32, 33, Section 6.5], and (ii) the projection sets computed by Lazard’s and McCallum’s projection operator are identical on more than 99.5% on all of **QF\_NRA** [33]. We argue, though, that working in

the domain of formal verification warrants the effort of obtaining a (provably) correct result, especially if it does not incur a performance overhead.

*Proof Generation.* Recently, generating formal proofs to certify the result of SMT solvers has become an area of focus. In particular, there is a large and ongoing effort to produce proofs in `cvc5`. The incremental linearization approach can be seen as an oracle which produces lemmas that are easy to prove individually, so `cvc5` does generate proofs for them; the complex part is finding those lemmas and making sure they actually help the solver make progress.

The situation is very different for cylindrical algebraic coverings: the produced lemma is the infeasible subset, and we usually have no simpler proof than the computations relying on CAD theory. That said, cylindrical algebraic coverings appear to be more amenable to automatic proof generation than traditional CAD-based approaches [4, 14]. In fact, although making these proofs detailed enough for automated verification is still an open problem, they are already broken into smaller parts that closely follow the tree-shaped computation of the algorithm. This allows `cvc5` to produce at least a proof skeleton in that case.

## 2.2 Incremental Linearization

Our theory solver for nonlinear (real) arithmetic optionally uses lemma schemas following the incremental linearization approaches described by Cimatti et al. [9] and Reynolds et al. [42]. These schemas incrementally refine candidate models from the linear arithmetic solver by introducing selected quantifier-free lemmas that express properties of multiplication, such as signedness (e.g.,  $x > 0 \wedge y > 0 \Rightarrow x \cdot y > 0$ ) or monotonicity (e.g.,  $|x| > |y| \Rightarrow x \cdot x > y \cdot y$ ). They are generated as needed to refute spurious models that violate these properties.

Most lemma schemas built-in in `cvc5` are crafted so as to avoid introducing new monomial terms or coefficients, since that could lead to non-termination in the CDCL(T) search. As a notable exception, we rely on a lemma schema for *tangent planes* for multiplication [9], which can be used to refute the candidate model for any application of the multiplication operator  $\cdot$  whose value in the linear model is inconsistent with the standard interpretation of  $\cdot$ . Note that since these lemmas depend upon the current model value chosen for arithmetic variables, tangent plane lemmas may introduce an unbounded number of new literals into the search. The set of lemma schemas used by the solver is user-configurable, as described in the following section.

## 2.3 Strategy

The overall theory solver for nonlinear arithmetic is built from several subsolvers, implementing the techniques described above, using a rather naive strategy, as summarized in Algorithm 1. After a spurious linear model has been constructed that cannot be repaired, we first apply a subset of the lemma schemas that do not introduce an unbounded number of new terms (with procedure `IncLinearizationLight`); then, we continue with the remaining lemma schemas

```

1 Function NlSolve(assertions)
2   if not LinearSolve(assertions) then return linear conflict
3   M = linearModel for assertions
4   if RepairModel(assertions, M) then return repaired model
5   if IncLinearizationLight(assertions, M) then return lemmas
6   if IncLinearizationFull(assertions, M) then return lemmas
7   return Coverings(assertions, M)

```

**Algorithm 1:** Strategy for nonlinear arithmetic solver

(with procedure `IncLinearizationFull`); finally, we resort to the coverings solver which is guaranteed to find either a conflict or a model. Internally, each procedure sequentially tries its assigned lemma schemas from [9, 42] until it constructs a lemma that can block the spurious model.

The approach is dynamically configured based on input options and the logic of the input formula. For example, by default, we disable `IncLinearizationFull` for `QF_NRA` as it tends to diverge in cases where the coverings solver quickly terminates.

## 2.4 Beyond QF\_NRA

The presented solver primarily targets quantifier-free nonlinear real arithmetic, but is used also in the presence of quantifiers and with multiple theories.

*Quantified Logics.* Solving quantified logics for nonlinear arithmetic requires solving quantifier-free subproblems, and thus any improvement to quantifier-free solving also benefits solving with quantifiers. In practice, however, the instantiation heuristics are just as important for overall solver performance.

*Multiple Theories.* The theory combination framework as implemented in `cvc5` requires evaluating equalities over the combined model. To support this functionality, real algebraic numbers had to be properly integrated into the entire solver; in particular, the ability to compute with these numbers could not be local to the cylindrical algebraic coverings module or even the nonlinear solver.

## 3 Experimental Results

We evaluate our implementation within `cvc5` (commit id 449dd7e) in comparison with other SMT solvers on all 11552 benchmarks in the quantifier-free nonlinear real arithmetic (`QF_NRA`) logic of SMT-LIB. We consider three configurations of `cvc5`, each of which runs a subset of steps from Algorithm 1. All the configurations run lines 2–4. In addition, `cvc5.cov` runs line 7, `cvc5.inclin` runs lines 5 and 6, and `cvc5` runs lines 5 and 7. All experiments were conducted on Intel Xeon E5-2637v4 CPUs with a time limit of 20 min and 8 GB memory.

We compare `cvc5` with recent versions of all other SMT solvers that participated in the `QF_NRA` logic of SMT-COMP 2021 [44]: `MathSAT` 5.6.6 [10], `SMT-RAT` 19.10.560 [13], `veriT` [7] (`veriT+raSAT+Redlog`), `Yices2` 2.6.4 [18] (`Yices-QS` for

quantified logics), and `z3` 4.8.14 [16]. `MathSAT` employs an abstraction-refinement mechanism very similar to the one described in Sect. 2.2; `veriT` [23] forwards nonlinear arithmetic problems to the external tools `raSAT` [45], which uses interval constraint propagation, and `Redlog/Reduce` [17], which focuses on virtual substitution and cylindrical algebraic decomposition; `SMT-RAT`, `Yices2`, and `z3` all implement some variant of MCSAT [30]. Note that `SMT-RAT` also implements the cylindrical algebraic coverings approach, but it is less effective than `SMT-RAT`'s adaptation of MCSAT [3].

				Beyond QF_NRA		sat unsat solved		
				NRA	Yices2	231	<b>3817</b>	<b>4048</b>
QF_NRA	sat	unsat	solved		z3	<b>236</b>	3812	<b>4048</b>
<code>cvc5</code>	<b>5137</b>	<b>5596</b>	<b>10733</b>		<code>cvc5.cov</code>	<b>236</b>	3809	4045
<code>Yices2</code>	4966	5450	10416		<code>cvc5</code>	221	3809	4030
<code>z3</code>	5136	5207	10343		<code>cvc5.inclin</code>	120	3786	3906
<code>cvc5.cov</code>	5001	5077	10078	QF_UFNRA	z3	<b>24</b>	<b>11</b>	<b>35</b>
<code>SMT-RAT</code>	4828	5038	9866		Yices2	23	<b>11</b>	34
<code>veriT</code>	4522	5034	9556		<code>cvc5</code>	20	<b>11</b>	31
<code>MathSAT</code>	3645	5357	9002		<code>cvc5.inclin</code>	12	<b>11</b>	23
<code>cvc5.inclin</code>	3421	5376	8797		<code>cvc5.cov</code>	2	<b>11</b>	13

(a)
(b)

**Fig. 2.** (a) Experiments for QF\_NRA (b) Experiments for NRA and QF\_UFNRA

Figure 2a shows that `cvc5` significantly outperforms all other QF\_NRA solvers. Both the coverings approach (`cvc5.cov`) and the incremental linearization approach (`cvc5.inclin`) contribute substantially to the overall performance of the unified solver in `cvc5`, with coverings solving many satisfiable instances, and incremental linearization helping on unsatisfiable ones. Even though `cvc5.inclin` closely follows [9], it outperforms `MathSAT` on unsatisfiable benchmarks, those where `cvc5` relies on incremental linearization the most.

Comparing `cvc5` and `Yices2` is particularly interesting, as the coverings approach in `cvc5` and the NLSAT solver in `Yices2` both rely on libpoly [29], thus using the same implementation of algebraic numbers and operations over them. Our integration of incremental linearization and algebraic coverings is compatible with the traditional CDCL(T) framework and outperforms the alternative NLSAT approach, which is specially tailored to nonlinear real arithmetic.

Going beyond QF\_NRA, we also evaluate the performance of our solver in the context of theory combination (with all 37 benchmarks from QF\_UFNRA) and quantifiers (with all 4058 benchmarks from NRA). There, `cvc5` is a close runner-up to `Yices2` and `z3`, thanks to the coverings subsolver which significantly improves `cvc5`'s performance. We conjecture that the remaining gap is due to components other than the nonlinear arithmetic solver, such as the solver for equality and uninterpreted functions, details of theory combination, or quantifier instantiation

heuristics. Interestingly, the sets of unsolved instances in **NRA** are almost disjoint for **cvc5.cov**, **Yices2** and **z3**, indicating that each tool could solve the remaining benchmarks with reasonable extra effort.

## 4 Conclusion

We have presented an approach for solving quantifier-free nonlinear real arithmetic problems that combines previous approaches based on incremental linearization [9] and cylindrical algebraic coverings [3] into one coherent abstraction-refinement loop. The resulting implementation is very effective, outperforming other state-of-the-art solver implementations, and integrates seamlessly in the CDCL(T) framework.

The general approach also applies to integer problems, quantified formulas, and instances with multiple theories, and can additionally be used in combination with transcendental functions [9] and bitwise conjunction for integers [47]. Further evaluations of these combinations are left to future work.

## References

1. Abbott, J., Bigatti, A.M., Palezzato, E.: New in CoCoA-5.2.4 and CoCoALib-0.99600 for SC-square. In: Satisfiability Checking and Symbolic Computation. CEUR Workshop Proceedings, vol. 2189, pp. 88–94. CEUR-WS.org (2018). <http://ceur-ws.org/Vol-2189/paper4.pdf>
2. Ábrahám, E., Corzilius, F., Johnsen, E.B., Kremer, G., Mauro, J.: Zephyrus2: on the fly deployment optimization using SMT and CP technologies. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) SETTA 2016. LNCS, vol. 9984, pp. 229–245. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-47677-3\\_15](https://doi.org/10.1007/978-3-319-47677-3_15)
3. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. *J. Logic. Algebr. Methods Program.* **119**(100633) (2021). <https://doi.org/10.1016/j.jlamp.2020.100633>
4. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Proving UNSAT in SMT: the case of quantifier free non-linear real arithmetic. *arXiv preprint arXiv:2108.05320* (2021)
5. Arnett, T.J., Cook, B., Clark, M., Rattan, K.: Fuzzy logic controller stability analysis using a satisfiability modulo theories approach. In: 19th AIAA Non-Deterministic Approaches Conference, p. 1773 (2017)
6. Basu, S., Pollack, R., Roy, M.F.: On the combinatorial and algebraic complexity of quantifier elimination. *J. ACM* **43**, 1002–1045 (1996). <https://doi.org/10.1145/235809.235813>
7. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: veriT: an open, trustworthy and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 151–156. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02959-2\\_12](https://doi.org/10.1007/978-3-642-02959-2_12)
8. Cashmore, M., Magazzeni, D., Zehtabi, P.: Planning for hybrid systems via satisfiability modulo theories. *J. Artif. Intell. Res.* **67**, 235–283 (2020). <https://doi.org/10.1613/jair.1.11751>

9. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Trans. Comput. Logic* **19**, 19:1–19:52 (2018). <https://doi.org/10.1145/3230639>
10. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) *TACAS 2013*. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-36742-7\\_7](https://doi.org/10.1007/978-3-642-36742-7_7)
11. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (ed.) *GI-Fachtagung 1975*. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975). [https://doi.org/10.1007/3-540-07407-4\\_17](https://doi.org/10.1007/3-540-07407-4_17)
12. Corzilius, F., Abraham, E.: Virtual substitution for SMT-solving. In: Owe, O., Steffen, M., Telle, J.A. (eds.) *FCT 2011*. LNCS, vol. 6914, pp. 360–371. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22953-4\\_31](https://doi.org/10.1007/978-3-642-22953-4_31)
13. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Abraham, E.: SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In: Heule, M., Weaver, S. (eds.) *SAT 2015*. LNCS, vol. 9340, pp. 360–368. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-24318-4\\_26](https://doi.org/10.1007/978-3-319-24318-4_26)
14. Davenport, J., England, M., Kremer, G., Tonks, Z., et al.: New opportunities for the formal proof of computational real geometry? *arXiv preprint arXiv:2004.04034* (2020)
15. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. *J. Symb. Comput.* **5**(1), 29–35 (1988). [https://doi.org/10.1016/S0747-7171\(88\)80004-X](https://doi.org/10.1016/S0747-7171(88)80004-X)
16. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) *TACAS 2008*. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_24](https://doi.org/10.1007/978-3-540-78800-3_24)
17. Dolzmann, A., Sturm, T.: REDLOG: computer algebra meets computer logic. *ACM SIGSAM Bull.* **31**(2), 2–9 (1997). <https://doi.org/10.1145/261320.261324>
18. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) *CAV 2014*. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08867-9\\_49](https://doi.org/10.1007/978-3-319-08867-9_49)
19. Ermon, S., Le Bras, R., Gomes, C.P., Selman, B., van Dover, R.B.: SMT-aided combinatorial materials discovery. In: Cimatti, A., Sebastiani, R. (eds.) *SAT 2012*. LNCS, vol. 7317, pp. 172–185. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31612-8\\_14](https://doi.org/10.1007/978-3-642-31612-8_14)
20. Fagerberg, R., Flamm, C., Merkle, D., Peters, P.: Exploring chemistry using SMT. In: Milano, M. (ed.) *CP 2012*. LNCS, pp. 900–915. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-33558-7\\_64](https://doi.org/10.1007/978-3-642-33558-7_64)
21. Faure, G., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: SAT modulo the theory of linear arithmetic: exact, inexact and commercial solvers. In: Kleine Büning, H., Zhao, X. (eds.) *SAT 2008*. LNCS, vol. 4996, pp. 77–90. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-79719-7\\_8](https://doi.org/10.1007/978-3-540-79719-7_8)
22. Fontaine, P., Ogawa, M., Sturm, T., Vu, X.T.: Subtropical satisfiability. In: Dixon, C., Finger, M. (eds.) *FroCoS 2017*. LNCS (LNAI), vol. 10483, pp. 189–206. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66167-4\\_11](https://doi.org/10.1007/978-3-319-66167-4_11)
23. Fontaine, P., Ogawa, M., Sturm, T., Vu, X.T., et al.: Wrapping computer algebra is surprisingly successful for non-linear SMT. In: *SC-square 2018-Third International Workshop on Satisfiability Checking and Symbolic Computation* (2018)
24. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) *CADE 2013*. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38574-2\\_14](https://doi.org/10.1007/978-3-642-38574-2_14)

25. Grigor'ev, D.Y., Vorobjov, N.: Solving systems of polynomial inequalities in subexponential time. *J. Symb. Comput.* **5**, 37–64 (1988). [https://doi.org/10.1016/S0747-7171\(88\)80005-1](https://doi.org/10.1016/S0747-7171(88)80005-1)
26. Hong, H.: An improvement of the projection operator in cylindrical algebraic decomposition. In: *International Symposium on Symbolic and Algebraic Computation*, pp. 261–264 (1990). <https://doi.org/10.1145/96877.96943>
27. Hong, H.: Comparison of several decision algorithms for the existential theory of the reals. RES report, Johannes Kepler University (1991)
28. Jovanović, D., Barrett, C., de Moura, L.: The design and implementation of the model constructing satisfiability calculus. In: *Formal Methods in Computer-Aided Design*, pp. 173–180. IEEE (2013). <https://doi.org/10.1109/FMCAD.2013.7027033>
29. Jovanovic, D., Dutertre, B.: LibPoly: a library for reasoning about polynomials. In: *Satisfiability Modulo Theories. CEUR Workshop Proceedings*, vol. 1889. CEUR-WS.org (2017). <http://ceur-ws.org/Vol-1889/paper3.pdf>
30. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) *IJCAR 2012. LNCS (LNAI)*, vol. 7364, pp. 339–354. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31365-3\\_27](https://doi.org/10.1007/978-3-642-31365-3_27)
31. Košta, M., Sturm, T.: A generalized framework for virtual substitution. *CoRR* abs/1501.05826 (2015)
32. Kremer, G.: Cylindrical algebraic decomposition for nonlinear arithmetic problems. Ph.D. thesis, RWTH Aachen University (2020). <https://doi.org/10.18154/RWTH-2020-05913>
33. Kremer, G., Brandt, J.: Implementing arithmetic over algebraic numbers: a tutorial for Lazard's lifting scheme in CAD. In: *Symbolic and Numeric Algorithms for Scientific Computing*, pp. 4–10 (2021). <https://doi.org/10.1109/SYNASC54541.2021.00013>
34. Kremer, G., Ábrahám, E.: Fully incremental cylindrical algebraic decomposition. *J. Symb. Comput.* **100**, 11–37 (2020). <https://doi.org/10.1016/j.jsc.2019.07.018>
35. Lazard, D.: An improved projection for cylindrical algebraic decomposition. In: Bajaj, C.L. (ed.) *Algebraic Geometry and Its Applications*, pp. 467–476. Springer, New York (1994). [https://doi.org/10.1007/978-1-4612-2628-4\\_29](https://doi.org/10.1007/978-1-4612-2628-4_29)
36. Loup, U., Scheibler, K., Corzilius, F., Ábrahám, E., Becker, B.: A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In: Bonacina, M.P. (ed.) *CADE 2013. LNCS (LNAI)*, vol. 7898, pp. 193–207. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38574-2\\_13](https://doi.org/10.1007/978-3-642-38574-2_13)
37. McCallum, S.: An improved projection operation for cylindrical algebraic decomposition. In: Caviness, B.F. (ed.) *EUROCAL 1985. LNCS*, vol. 204, pp. 277–278. Springer, Heidelberg (1985). [https://doi.org/10.1007/3-540-15984-3\\_277](https://doi.org/10.1007/3-540-15984-3_277)
38. McCallum, S., Parusiński, A., Paunescu, L.: Validity proof of Lazard's method for cad construction. *J. Symb. Comput.* **92**, 52–69 (2019). <https://doi.org/10.1016/j.jsc.2017.12.002>
39. de Moura, L., Jovanović, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) *VMCAI 2013. LNCS*, vol. 7737, pp. 1–12. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-35873-9\\_1](https://doi.org/10.1007/978-3-642-35873-9_1)
40. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *J. ACM* **53**(6), 937–977 (2006). <https://doi.org/10.1145/1217856.1217859>
41. Renegar, J.: A faster PSPACE algorithm for deciding the existential theory of the reals. In: *Symposium on Foundations of Computer Science*, pp. 291–295 (1988). <https://doi.org/10.1109/SFCS.1988.21945>



42. Reynolds, A., Tinelli, C., Jovanović, D., Barrett, C.: Designing theory solvers with extensions. In: Dixon, C., Finger, M. (eds.) *FroCoS 2017*. LNCS (LNAI), vol. 10483, pp. 22–40. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66167-4\\_2](https://doi.org/10.1007/978-3-319-66167-4_2)
43. Roselli, S.F., Bengtsson, K., Åkesson, K.: SMT solvers for job-shop scheduling problems: models comparison and performance evaluation. In: *International Conference on Automation Science and Engineering (CASE)*, pp. 547–552 (2018). <https://doi.org/10.1109/COASE.2018.8560344>
44. SMT-COMP 2021 (2021). <https://smt-comp.github.io/2021/>
45. Tung, V.X., Van Khanh, T., Ogawa, M.: raSAT: an SMT solver for polynomial constraints. In: Olivetti, N., Tiwari, A. (eds.) *IJCAR 2016*. LNCS (LNAI), vol. 9706, pp. 228–237. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-40229-1\\_16](https://doi.org/10.1007/978-3-319-40229-1_16)
46. Weispfenning, V.: Quantifier elimination for real algebra—the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.* **8**(2), 85–101 (1997). <https://doi.org/10.1007/s002000050055>
47. Zohar, Y., et al.: Bit-precise reasoning via Int-blasting. In: Finkbeiner, B., Wies, T. (eds.) *VMCAI 2022*. LNCS, vol. 13182, pp. 496–518. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-94583-1\\_24](https://doi.org/10.1007/978-3-030-94583-1_24)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.







# Preprocessing of Propagation Redundant Clauses

Joseph E. Reeves<sup>(✉)</sup>, Marijn J. H. Heule, and Randal E. Bryant

Carnegie Mellon University, Pittsburgh, PA, USA  
{jereeves, mheule, randy.bryant}@cs.cmu.edu

**Abstract.** The *propagation redundant* (PR) proof system generalizes the *resolution* and *resolution asymmetric tautology* proof systems used by *conflict-driven clause learning* (CDCL) solvers. PR allows short proofs of unsatisfiability for some problems that are difficult for CDCL solvers. Previous attempts to automate PR clause learning used hand-crafted heuristics that work well on some highly-structured problems. For example, the solver SADICAL incorporates PR clause learning into the CDCL loop, but it cannot compete with modern CDCL solvers due to its fragile heuristics. We present PRELEARN, a preprocessing technique that learns short PR clauses. Adding these clauses to a formula reduces the search space that the solver must explore. By performing PR clause learning as a preprocessing stage, PR clauses can be found efficiently without sacrificing the robustness of modern CDCL solvers. On a large portion of SAT competition benchmarks we found that preprocessing with PRELEARN improves solver performance. In addition, there were several satisfiable and unsatisfiable formulas that could only be solved after preprocessing with PRELEARN. PRELEARN supports proof logging, giving a high level of confidence in the results.

## 1 Introduction

*Conflict-driven clause learning* (CDCL) [27] is the standard paradigm for solving the satisfiability problem (SAT) in propositional logic. CDCL solvers learn clauses implied through *resolution* inferences. Additionally, all competitive CDCL solvers use pre- and in-processing techniques captured by the *resolution asymmetric tautology* (RAT) proof system [21]. As examples, the well-studied pigeonhole and mutilated chessboard problems are challenging benchmarks with exponentially-sized resolution proofs [1, 12]. It is possible to construct small hand-crafted proofs for the pigeonhole problem using *extended resolution* (ER) [8], a proof system that allows the introduction of new variables [32]. ER can be expressed in RAT but has proved difficult to automate due to the large search space. Even with modern inprocessing techniques, many CDCL solvers struggle on these seemingly simple problems. The *propagation redundant* (PR) proof system allows short proofs for these problems [14, 15], and unlike in ER, no new variables are required. This makes PR an attractive candidate for automation.

At a high level, CDCL solvers make decisions that typically yield an unsatisfiable branch of a problem. The clause that prunes the unsatisfiable branch from the search space is learned, and the solver continues by searching another branch. PR extends this

---

The authors are supported by the NSF under grant CCF-2108521.

© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 106–124, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_8](https://doi.org/10.1007/978-3-031-10769-6_8)

paradigm by allowing more aggressive pruning. In the PR proof system a branch can be pruned as long as there exists another branch that is at least as satisfiable. As an example, consider the mutilated chessboard. The mutilated chessboard problem involves finding a covering of  $2 \times 1$  dominos on an  $n \times n$  chessboard with two opposite corners removed (see Section 5.4). Given two horizontally oriented dominos covering a  $2 \times 2$  square, two vertically oriented dominos could cover the same  $2 \times 2$  square. For any solution that uses the dominos in the horizontal orientation, replacing them with the dominos in the vertical orientation would also be a solution. The second orientation is as satisfiable as the first, and so the first can be pruned from the search space. Even though the number of possible solutions may be reduced, the pruning is satisfiability preserving. This is a powerful form of reasoning that can efficiently remove many symmetries from the mutilated chessboard, making the problem much easier to solve [15].

The *satisfaction-driven clause learning* (SDCL) solver SADICAL [16] incorporates PR clause learning into the CDCL loop. SADICAL implements hand-crafted decision heuristics that exploit the canonical structure of the pigeonhole and mutilated chessboard problems to find short proofs. However, SADICAL’s performance deteriorates under slight variations to the problems including different constraint encodings [7]. The heuristics were developed from a few well-understood problems and do not generalize to other problem classes. Further, the heuristics for PR clause learning are likely ill-suited for CDCL, making the solver less robust.

In this paper, we present PRELEARN, a preprocessing technique for learning PR clauses. PRELEARN alternates between finding and learning PR clauses. We develop multiple heuristics for finding PR clauses and multiple configurations for learning some subset of the found PR clauses. As PR clauses are learned we use failed literal probing [11] to find unit clauses implied by the formula. The preprocessing is made efficient by taking advantage of the inner/outer solver framework in SADICAL. The learned PR clauses are added to the original formula, aggressively pruning the search space in an effort to guide CDCL solvers to short proofs. With this method PR clauses can be learned without altering the complex heuristics that make CDCL solvers robust. PRELEARN focuses on finding short PR clauses and failed literals to effectively reduce the search space. This is done with general heuristics that work across a wide range of problems.

Most SAT solvers support logging proofs of unsatisfiability for independent checking [17, 20, 33]. This has proved valuable for verifying solutions independent of a (potentially buggy) solver. Modern SAT solvers log proofs in the DRAT proof system (RAT [21] with deletions). DRAT captures all widely used pre- and in-processing techniques including bounded variable elimination [10], bounded variable addition [26], and extended learning [4, 32]. DRAT can express the common symmetry-breaking techniques, but it is complicated [13]. PR can compactly express some symmetry-breaking techniques [14, 15], yielding short proofs that can be checked by the proof checker DPR-TRIM [16]. PR gives a framework for strong symmetry-breaking inferences and also maintains the highly desirable ability to independently verify proofs.

The contributions of this paper include: (1) giving a high-level algorithm for extracting PR clauses, (2) implementing several heuristics for finding and learning PR clauses, (3) evaluating the effectiveness of different heuristic configurations, and (4) assessing the impact of PRELEARN on solver performance. PRELEARN improves the

performance of the CDCL solver KISSAT on a quarter of the satisfiable and unsatisfiable competition benchmarks we considered. The improvement is significant for a number of instances that can only be solved by KISSAT after preprocessing. Most of them come from hard combinatorial problems with small formulas. In addition, PRELEARN directly produces refutation proofs for the mutilated chessboard problem containing only unit and binary PR clauses.

## 2 Preliminaries

We consider propositional formulas in *conjunctive normal form* (CNF). A CNF formula  $\psi$  is a conjunction of *clauses* where each clause is a disjunction of *literals*. A literal  $l$  is either a variable  $x$  (positive literal) or a negated variable  $\bar{x}$  (negative literal). For a set of literals  $L$  the formula  $\psi(L)$  is the clauses  $\{C \in \psi \mid C \cap L \neq \emptyset\}$ .

An *assignment* is a mapping from variables to truth values 1 (*true*) and 0 (*false*). An assignment is *total* if it assigns every variable to a value, and *partial* if it assigns a subset of variables to values. The set of variables occurring in a formula, assignment, or clause is given by  $\text{var}(\psi)$ ,  $\text{var}(\alpha)$ , or  $\text{var}(C)$ . For a literal  $l$ ,  $\text{var}(l)$  is a variable.

An assignment  $\alpha$  *satisfies* a positive (negative) literal  $l$  if  $\alpha$  maps  $\text{var}(l)$  to true ( $\alpha$  maps  $\text{var}(l)$  to false, respectively), and *falsifies* it if  $\alpha$  maps  $\text{var}(l)$  to false ( $\alpha$  maps  $\text{var}(l)$  to true, respectively). We write a finite partial assignment as the set of literals it satisfies. An assignment satisfies a clause if the clause contains a literal satisfied by the assignment. An assignment satisfies a formula if every clause in the formula is satisfied by the assignment. A formula is *satisfiable* if there exists a satisfying assignment, and *unsatisfiable* otherwise. Two formula are *logically equivalent* if they share the same set of satisfying assignments. Two formulas are *satisfiability equivalent* if they are either both satisfiable or both unsatisfiable.

If an assignment  $\alpha$  satisfies a clause  $C$  we define  $C|_\alpha = \top$ , otherwise  $C|_\alpha$  represents the clause  $C$  with the literals falsified by  $\alpha$  removed. The empty clause is denoted by  $\perp$ . The formula  $\psi$  reduced by an assignment  $\alpha$  is given by  $\psi|_\alpha = \{C|_\alpha \mid C \in \psi \text{ and } C|_\alpha \neq \top\}$ . Given an assignment  $\alpha = l_1 \dots l_n$ ,  $C = (\bar{l}_1 \vee \dots \vee \bar{l}_n)$  is the clause that *blocks*  $\alpha$ . The assignment *blocked* by a clause is the negation of the literals in the clause. The literals touched by an assignment is defined by  $\text{touched}_\alpha(C) = \{l \mid l \in C \text{ and } \text{var}(l) \in \text{var}(\alpha)\}$  for a clause. For a formula  $\psi$ ,  $\text{touched}_\alpha(\psi)$  is the union of touched variables for each clause in  $\psi$ . A *unit* is a clause containing a single literal. The *unit clause rule* takes the assignment  $\alpha$  of all units in a formula  $\psi$  and generates  $\psi|_\alpha$ . Iteratively applying the unit clause rule until fixpoint is referred to as *unit propagation*. In cases where unit propagation yields  $\perp$  we say it derived a *conflict*. A formula  $\psi$  *implies* a formula  $\psi'$ , denoted  $\psi \models \psi'$ , if every assignment satisfying  $\psi$  satisfies  $\psi'$ . By  $\psi \vdash_1 \psi'$  we denote that for every clause  $C \in \psi'$ , applying unit propagation to the assignment blocked by  $C$  in  $\psi$  derives a conflict. If unit propagation derives a conflict on the formula  $\psi \cup \{\{l\}\}$ , we say  $l$  is a *failed literal* and the unit  $\bar{l}$  is logically implied by the formula. Failed literal probing [11] is the process of successively assigning literals to check if units are implied by the formula. In its simplest form, probing involves assigning a literal  $l$  and learning the unit  $\bar{l}$  if unit propagation derives a conflict, otherwise  $l$  is unassigned and the next literal is checked.

To evaluate the satisfiability of a formula, a CDCL solver [27] iteratively performs the following operations: First, the solver performs unit propagation, then tests for a conflict. Unit propagation is made efficient with two-literal watch pointers [28]. If there is no conflict and all variables are assigned, the formula is satisfiable. Otherwise, the solver chooses an unassigned variable through a variable decision heuristic [6, 25], assigns a truth value to it, and performs unit propagation. If, however, there is a conflict, the solver performs conflict analysis potentially learning a short clause. In case this clause is the empty clause, the formula is unsatisfiable.

### 3 The PR Proof System

A clause  $C$  is *redundant* w.r.t. a formula  $\psi$  if  $\psi$  and  $\psi \cup \{C\}$  are *satisfiability equivalent*. The clause sequence  $\psi, C_1, C_2, \dots, C_n$  is a *clausal proof* of  $C_n$  if each clause  $C_i$  ( $1 \leq i \leq n$ ) is redundant w.r.t.  $\psi \cup \{C_1, C_2, \dots, C_{i-1}\}$ . The proof is a *refutation* of  $\psi$  if  $C_n$  is  $\perp$ . Clausal proof systems may also allow deletion. In a refutation proof clauses can be deleted freely because the deletion cannot make a formula less satisfiable.

Clausal proof systems are distinguished by the kinds of redundant clauses they allow to be added. The standard SAT solving paradigm CDCL learns clauses implied through *resolution*. These clauses are logically implied by the formula, and fall under the *reverse unit propagation* (RUP) proof system. The *Resolution Asymmetric Tautology* (RAT) proof system generalizes RUP. All commonly used inprocessing techniques emit DRAT proofs. The *propagation redundant* (PR) proof system generalizes RAT by allowing the pruning of branches *without loss of satisfaction*.

Let  $C$  be a clause in the formula  $\psi$  and  $\alpha$  the assignment blocked by  $C$ . Then  $C$  is PR w.r.t.  $\psi$  if and only if there exists an assignment  $\omega$  such that  $\psi|_{\alpha} \vdash_1 \psi|_{\omega}$  and  $\omega$  satisfies  $C$ . Intuitively, this allows inferences that block a partial assignment  $\alpha$  as long as another assignment  $\omega$  is as satisfiable. This means every assignment containing  $\alpha$  that satisfies  $\psi$  can be transformed to an assignment containing  $\omega$  that satisfies  $\psi$ .

Clausal proofs systems must be checkable in polynomial time to be useful in practice. RUP and RAT are efficiently checkable due to unit propagation. In general, determining if a clause is PR is an NP-complete problem [18]. However, a PR proof is checkable in polynomial time if the witness assignments  $\omega$  are included. A clausal proof with witnesses will look like  $\psi, (C_1, \omega_1), (C_2, \omega_2), \dots, (C_n, \omega_n)$ . The proof checker DPR-TRIM can efficiently check PR proofs that include witnesses. Further, DPR-TRIM can emit proofs in the LPR format. They can be validated by the formally-verified checker CAKE-LPR [31], which was used to validate results in recent SAT competitions.

### 4 Pruning Predicates and SADICAL

Determining if a clause is PR is NP-complete and can naturally be formulated in SAT. Given a clause  $C$  and formula  $\psi$ , a *pruning predicate* is a formula such that if it is satisfiable, the clause  $C$  is redundant w.r.t.  $\psi$ . SADICAL uses two pruning predicates to determine if a clause is PR: *positive reduct* and *filtered positive reduct*. If either predicate is satisfiable, the satisfying assignment serves as the witness showing the clause is PR.

Given a formula  $\psi$  and assignment  $\alpha$ , the *positive reduct* is the formula  $G \wedge C$  where  $C$  is the clause that blocks  $\alpha$  and  $G = \{\text{touched}_\alpha(D) \mid D \in \psi \text{ and } D|_\alpha = \top\}$ . If the positive reduct is satisfiable, the clause  $C$  is PR w.r.t.  $\psi$ . The positive reduct is satisfiable iff the clause blocked by  $\alpha$  is a *set-blocked* clause [23].

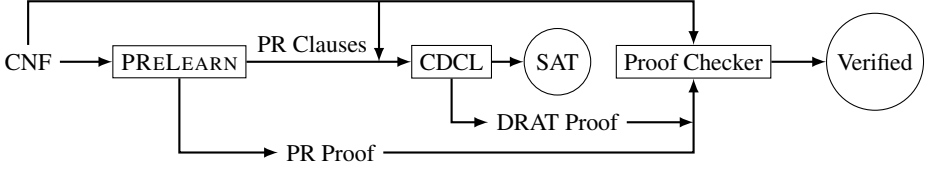
Given a formula  $\psi$  and assignment  $\alpha$ , the *filtered positive reduct* is the formula  $G \wedge C$  where  $C$  is the clause that blocks  $\alpha$  and  $G = \{\text{touched}_\alpha(D) \mid D \in \psi \text{ and } D|_\alpha \not\models_1 \text{touched}_\alpha(D)\}$ . If the filtered positive reduct is satisfiable, the clause  $C$  is PR w.r.t.  $\psi$ . The filtered positive reduct is a subset of the positive reduct and is satisfiable iff the clause blocked by  $\alpha$  is a *set-propagation redundant* clause [14]. Example 1 shows a formula for which the positive and filtered positive reducts are different, and only the filtered positive reduct is satisfiable.

*Example 1.* Given the formula  $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2)$ , the positive reduct with  $\alpha = x_1$  is  $(x_1) \wedge (\bar{x}_1)$ , which is unsatisfiable. The clause  $(x_1)$  can be filtered, giving the filtered positive reduct  $(\bar{x}_1)$ , which is satisfiable.

SADICAL [16] uses satisfaction-driven clause learning (SDCL) that extends CDCL by learning PR clauses [18] based on (filtered) positive reducts. SADICAL uses an inner/outer solver framework. The outer solver attempts to solve the SAT problem with SDCL. SDCL diverges from the basic CDCL loop when unit propagation after a decision does not derive a conflict. In this case a reduct is generated using the current assignment, and the inner solver attempts to solve the reduct using CDCL. If the reduct is satisfiable, the PR clause blocking the current assignment is learned, and the SDCL loop continues. The PR clause can be simplified by removing all non-decision variables from the assignment. SADICAL emits PR proofs by logging the satisfying assignment of the reduct as the witness, and these proofs are verified with DPR-TRIM. The key to SADICAL finding good PR clauses leading to short proofs is the decision heuristic, because variable selection builds the candidate PR clauses. Hand-crafted decision heuristics enable SADICAL to find short proofs on pigeonhole and mutilated chessboard problems. However, these heuristics differ significantly from the score-based heuristics in most CDCL solvers. Our experiences with SaCiDaL suggest that improving the heuristics for SDCL reduces the performance of CDCL and the other way around. This may explain why SADICAL performs worse than standard CDCL solvers on the majority of the SAT competition benchmarks. While SADICAL integrates finding PR clauses of arbitrary size in the main search loop, our tool focuses on learning short PR clauses as a preprocessing step. This allows us to develop good heuristics for PR learning without compromising the main search loop.

## 5 Extracting PR Clauses

The goal of PRELEARN is to find useful PR clauses that improve the performance of CDCL solvers on both satisfiable and unsatisfiable instances. Figure 1 shows how a SAT problem is solved using PRELEARN. For some preset time limit, PR clauses are found and then added to the original formula. Interleaved in this process is failed literal probing to check if unit clauses can be learned. When the preprocessing stage ends, the new formula that includes learned PR clauses is solved by a CDCL solver. If the



**Fig. 1.** Solving a formula with PRELEARN and a CDCL solver.

formula is satisfiable, the solver will produce a satisfying assignment. If the formula is unsatisfiable, a refutation proof of the original formula can be computed by combining the satisfaction preserving proof from PRELEARN and the refutation proof emitted by the CDCL solver. The complete proof can be verified with DPR-TRIM.

PRELEARN alternates between finding PR clauses and learning PR clauses. Candidate PR clauses are found by iterating over each variable in the formula, and for each variable constructing clauses that include that variable. To determine if a clause is PR, the positive reduct generated by that clause is solved. It can be costly to generate and solve many positive reducts, so heuristics are used to find candidate clauses that are more likely to be PR. It is possible to find multiple PR clauses that conflict with each other. PR clauses are conflicting if adding one of the PR clauses to the formula makes the other no longer PR. Learning PR clauses involves selecting PR clauses that are non-conflicting. The selection may maximize the number of PR clauses learned or optimize for some other metric. Adding PR clauses and units derived from probing may cause new clauses to become PR, so the entire process is iterated multiple times.

## 5.1 Finding PR Clauses

PR clauses are found by constructing a set of candidate clauses and solving the positive reduct generated by each clause. In SADICAL the candidates are the clauses blocking the partial assignment of the solver after each decision in the SDCL loop that does not derive a conflict. In effect, candidates are constructed using the solver's variable decision heuristic. We take a more general approach, constructing sets of candidates for each variable based on unit propagation and the partial assignment's neighbors.

For a variable  $x$ ,  $\text{neighbors}(x)$  denotes the set of variables occurring in clauses containing literal  $x$  or  $\bar{x}$ , excluding variable  $x$ . For a partial assignment  $\alpha$ ,  $\text{neighbors}(\alpha)$  denotes  $\bigcup_{x \in \text{var}(\alpha)} \text{neighbors}(x) \setminus \text{var}(\alpha)$ . Candidate clauses for a literal  $l$  are generated in the following way:

- Let  $\alpha$  be the partial assignment found by unit propagation starting with the assignment that makes  $l$  true.
- Generate the candidate PR clauses  $\{(\bar{l} \vee y), (\bar{l} \vee \bar{y}) \mid y \in \text{neighbors}(\alpha)\}$ .

Example 2 shows how candidate binary clauses are constructed using both polarities of an initial variable  $x$ . In Example 3 the depth is expanded to reach more variables and create larger sets of candidate clauses. The depth parameter is used in Section 5.4.

*Example 2.* Consider the following formula:  $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_6 \vee x_7) \wedge (x_3 \vee x_7 \vee x_8) \wedge (x_8 \vee x_9)$ ,

**Case 1:** We start with  $\text{var}(x_1) = 1$  and perform unit propagation resulting in  $\alpha = \{x_1 x_3\}$ . Observe that  $\text{neighbors}(\alpha) = \{x_2, x_4, x_5, x_7, x_8\}$ . The generated candidate clauses are  $(\bar{x}_1 \vee x_2)$ ,  $(\bar{x}_1 \vee \bar{x}_2)$ ,  $(\bar{x}_1 \vee x_4)$ ,  $(\bar{x}_1 \vee \bar{x}_4)$ ,  $\dots$ ,  $(\bar{x}_1 \vee x_8)$ ,  $(\bar{x}_1 \vee \bar{x}_8)$ .

**Case 2:** We start with  $\text{var}(x_1) = 0$  and perform unit propagation resulting in  $\alpha = \{\bar{x}_1 \bar{x}_2\}$ . Observe that  $\text{neighbors}(\alpha) = \{x_3, x_4, x_5, x_6, x_7\}$ . The generated candidate clauses are  $(x_1 \vee x_3)$ ,  $(x_1 \vee \bar{x}_3)$ ,  $(x_1 \vee x_4)$ ,  $(x_1 \vee \bar{x}_4)$ ,  $\dots$ ,  $(x_1 \vee x_7)$ ,  $(x_1 \vee \bar{x}_7)$ .

*Example 3.* Take the formula from Example 2 and assignment of  $\text{var}(x_1) = 1$  as in case 1. The set of candidate clauses can be expanded by also considering the unassigned neighbors of the variables in  $\text{neighbors}(\alpha)$ . For example,  $\text{neighbors}(x_8) = \{x_3, x_7, x_9\}$ , of which  $x_9$  is new and unassigned. This adds  $(\bar{x}_1 \vee x_9)$  and  $(\bar{x}_1 \vee \bar{x}_9)$  to the set of candidate clauses. This can be iterated by including neighbors of new unassigned variables from the prior step.

We consider both polarities when constructing candidates for a variable. After all candidates for a variable are constructed, the positive reduct for each candidate is generated and solved in order. Note that propagated literals appearing in the partial assignment do not appear in the PR clause. The satisfying assignment is stored as the witness and the PR clause may be learned immediately depending on the learning configuration.

This process is naturally extended to ternary clauses. The binary candidates are generated, and for each candidate  $(x \vee y)$ ,  $x$  and  $y$  are assigned to false in the first step. The variables  $z \in \text{neighbors}(\alpha)$  yield clauses  $(x \vee y \vee z)$  and  $(x \vee y \vee \bar{z})$ . This approach can generate many candidate ternary clauses depending on the connectivity of the formula since each candidate binary clause is expanded. A filtering operation would be useful to avoid the blow-up in number of candidates. There are likely diminishing returns when searching for larger PR clauses because (1) there are more possible candidates, (2) the positive reducts are likely larger, and (3) each clause blocks less of the search space. We consider only unit and binary candidate clauses in our main evaluation.

Ideally, we should construct candidate clauses that are likely PR to reduce the number of failed reducts generated. Note, the (filtered) positive reduct can only be satisfiable if given the partial assignment there exists a reduced, satisfied clause. By focusing on neighbors, we guarantee that such a clause exists. The *reduced* heuristic in SADICAL finds variables in all reduced but unsatisfied clauses. The idea behind this heuristic is to direct the assignment towards conditional autarkies that imply a satisfiable positive reduct [18]. The neighbors approach generalizes this to variables in all reduced clauses whether or not they are unsatisfied. A comparison can be found in our repository.

## 5.2 Learning PR Clauses

Given multiple clauses that are PR w.r.t. the same formula, it is possible that some of the clauses conflict with each other and cannot be learned simultaneously. Example 4 shows how learning one PR clause may invalidate the witness of another PR clause. It may be that a different witness exists, but finding it requires regenerating the positive reduct to include the learned PR clause and solving it. The simplest way to avoid conflicting PR clause is to learn PR clauses as they are found. When a reduct is satisfiable,



the PR clauses is added to the formula and logged with its witness in the proof. Then subsequent reducts will be generated from the formula including all added PR clauses. Therefore, a satisfiable reduct ensures a PR clause can be learned.

Alternatively, clauses can be found in batches, then a subset of nonconflicting clauses can be learned. The set of conflicts between PR clauses can be computed in polynomial time. For each pair of PR clauses  $C$  and  $D$ , if the assignment that generated the pruning predicate for  $D$  touches  $C$  and  $C$  is not satisfied by the witness of  $D$ , then  $C$  conflicts with  $D$ . In some cases reordering the two PR clauses may avoid a conflict. In Example 4 learning the second clause would not affect the validity of the first clauses' witness. Once the conflicts are known, clauses can be learned based on some heuristic ordering. Batch learning configurations are discussed more in the following section.

*Example 4.* Assume the following clause witness pairs are valid in a formula  $\psi$ :  $\{(x_1 \vee x_2 \vee x_3), x_1 \bar{x}_2 \bar{x}_3\}$ , and  $\{(x_1 \vee \bar{x}_2 \vee x_4), \bar{x}_1 \bar{x}_2 x_4\}$ . The first clause conflicts with the second. If the first clause is added to  $\psi$ , the clause  $(x_1 \vee x_2)$  would be in the positive reduct for the second clause, but it is not satisfied by the witness of the second clause.

### 5.3 Additional Configurations

The sections above describe the PRELEARN configuration used in the main evaluation, i.e., finding candidate PR clauses with the neighbors heuristic and learning clauses instantly as the positive reducts are solved. In this section we present several additional configurations. The time-constrained reader may skip ahead to Section 5.4 for the presentation of our main results.

In batch learning a set of PR clauses are found in batches then learned. Learning as many nonconflicting clauses as possible coincides with the maximum independent set problem. This problem is NP-Hard. We approximate the solution by adding the clause causing the fewest conflicts with unblocked clauses. When a clause is added, the clauses it blocks are removed from the batch and conflict counts are recalculated. Alternatively, clauses can be added in a random order. Random ordering requires less computation at the cost of potentially fewer learned PR clauses.

The neighbors heuristic for constructing candidate clauses can be modified to include a depth parameter.  $\text{neighbors}(i)$  indicates the number of iterations expanding the variables. For example,  $\text{neighbors}(2)$  expands on the variables in  $\text{neighbors}(1)$ , seen in Example 3. We also implement the reduced heuristic, shown in Example 5. Detailed evaluations and comparisons can be found in our repository. In general, we found that the additional configurations did not improve on our main configuration. More work needs to be done to determine when and how to apply these additional configurations.

*Example 5.* Given the set of clauses  $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (x_3 \vee x_5)$ , and initial assignment  $\alpha = x_1$ , only the second clause is reduced and not satisfied, giving  $\text{reduced}(\alpha) = \{x_3, x_4\}$  and candidate clauses  $(\bar{x}_1 \vee x_3)$ ,  $(\bar{x}_1 \vee x_4)$ ,  $(\bar{x}_1 \vee \bar{x}_3)$ ,  $(\bar{x}_1 \vee \bar{x}_4)$ .

### 5.4 Implementation

PRELEARN was implemented using the inner/outer-solver framework in SADICAL. The inner solver acts the same as in SADICAL, solving pruning predicates using CDCL.

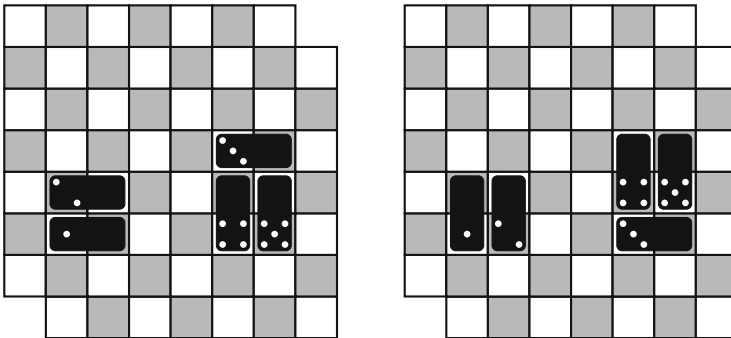


The outer solver is not used for SDCL, but the SDCL data-structures are used to find and learn PR clauses. The outer solver is initialized with the original formula and maintains the list of variables, clauses, and watch pointers. By default, the outer solver has no variables assigned other than learned units. When finding candidates, the variables in the partial clause are assigned in the outer solver. Unit propagation makes it possible to find all reduced clauses in the formula with a single pass. This is necessary for constructing the positive reduct. After a candidate clause has been assigned and the positive reduct solved, the variables are unassigned. This returns the outer solver to the top-level before examining the next candidate. When a PR clause is learned, it is added to the formula along with its watch pointers. Additionally, failed literals are found if assigning a variable at the top-level causes a conflict through unit propagation. The negation of a failed literal is a unit that can be added to the formula.

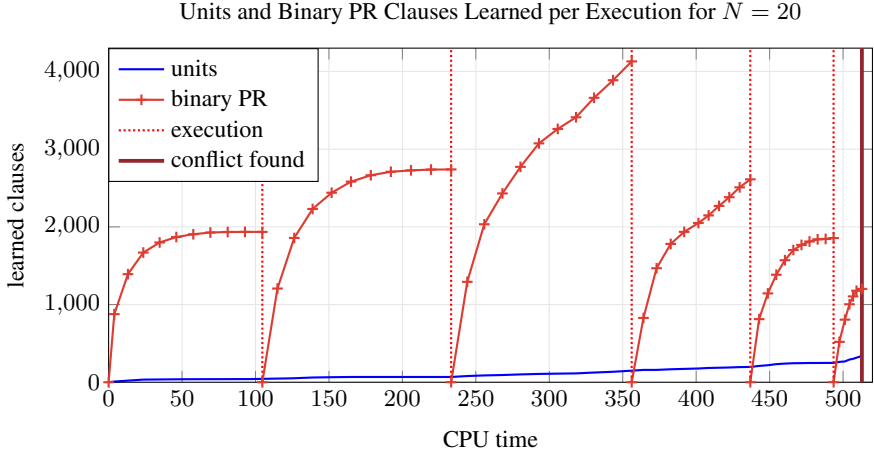
In a single iteration each variable in the formula is processed in a breadth-first search (BFS) starting from the first variable in the numbering. When a variable is encountered it is first checked whether either assignment of the variable is a failed literal or a unit PR clause. If not, binary candidates are generated based on the selected heuristic and PR clauses are learned based on the learning configuration. Variables are added to the frontier of the BFS as they are encountered during candidate clause generation, but they are not repeated. Optionally, after all variables have been encountered the BFS restarts, now constructing ternary candidates. The repetition continues to the desired clause length. Then another iteration begins again with binary clauses. Running PRELEARN multiple times is important because adding PR clauses in one iteration may allow additional clauses to be added in the next.

## 6 Mutilated Chessboard

The *mutilated chessboard* is an  $n \times n$  grid of alternating black and white squares with two opposite corners removed. The problem is whether or not the board can be covered with  $2 \times 1$  dominoes. This can be encoded in CNF by using variables to represent



**Fig. 2.** Occurrences of two horizontal dominoes may be replaced by two vertical dominoes in a solution. Similarly, occurrences of a horizontal domino atop two vertical dominoes can be replaced by shifting the horizontal domino down.



**Fig. 3.** Unit and binary PR clauses learned each execution (red-dotted line) until a contradiction was found. Markers on binary PR lines represent an iteration within an execution.

domino placements on the board. At-most-one constraints (using the pairwise encodings) say only one domino can cover each square, and at-least-one constraints (using a disjunction) say some domino must cover each square.

In recent SAT competitions, no proof-generating SAT solver could deal with instances larger than  $N = 18$ . In ongoing work, we found refutation proofs that contain only units and binary PR clauses for some boards of size  $N \leq 30$ . PRELEARN can be modified to automatically find proofs of this type. Running iterations of PRELEARN until *saturation*, meaning no new binary PR clauses or units can be found, yields some set of units and binary PR clauses. Removing the binary PR clauses from the formula and rerunning PRELEARN will yield additional units and a new set of binary PR clauses. Repeating the process of removing binary PR clauses and keeping units will eventually derive the empty clause for this problem. Figure 3 gives detailed values for  $N = 20$ . Within each execution (red dotted lines) there are at most 10 iterations (red tick markers), and each iteration learns some set of binary PR clauses (red). Some executions saturate binary PR clauses before the tenth iteration and exit early. At the end of each execution the binary PR clauses are deleted, but the units (blue) are kept for the following execution. A complete DPR proof (PR with deletion) can be constructed by adding deletion information for the binary PR clauses removed between each execution when concatenating the PRELEARN proofs. The approach works for mutilated chess because in each execution there are many binary PR clauses that can be learned and will lead to units, but they are mutually exclusive and cannot be learned simultaneously. Further, adding units allows new binary PR clauses to be learned in following executions.

Table 1 shows the statistics for PRELEARN. Achieving these results required some modifications to the configuration of PRELEARN. First, notice in Figure 2 the PR clauses that can be learned involve blocking one domino orientation that can be replaced by a symmetric orientation. To optimize for these types of PR clauses, we only

**Table 1.** Statistics running multiple executions of PRELEARN on the mutilated chessboard problem with the configurations described below. Total units includes failed literals and learned PR units. The average units and average binary PR clauses learned during each execution (Exe.) are shown as well.

$N$	Time (s)	# Exe.	Avg. (s)	Total Units	Total Bin.	Avg. Units	Avg. Bin.
8	0.14	1	0.14	30	164	30.00	164.00
12	4.94	1	4.94	103	1,045	103.00	1,045.00
16	62.47	2	31.23	195	3,988	97.50	1,994.00
20	513.12	6	85.52	339	1,4470	56.50	2,411.67
24	4,941.38	26	190.05	512	64,038	19.69	2,463.00

constructed candidates where the first literal was negative. The neighbors heuristic had to be increased to a depth of 6, meaning more candidates were generated for each variable. Intuitively, the proof is constructed by adding binary PR clauses in order to find negative units (dominos that cannot be placed) around the borders of the board. Following iterations build more units inwards, until a point is reached where units cover almost the entire board. This forces an impossible domino placement leading to a contradiction. Complete proofs using only units and binary PR clauses were found for boards up to size  $N = 24$  within 5,000 seconds. We verified all proofs using DPR-TRIM. The mutilated chessboard has a high degree of symmetry and structure, making it suitable for this approach. For most problems it is not expected that multiple executions while keeping learned units will find new PR clauses.

Experiments were done with several configurations (see Section 5.3) to find the best results. We found that increasing the depth of neighbors was necessary for larger boards including  $N = 24$ . Increasing the depth allows more binary PR clauses to be found, at the cost of generating more reducts. This is necessary to find units. The reduced heuristic (a subset of neighbors) did not yield complete proofs. We also tried incrementing the depth after each execution starting with 1 and resetting at 9. In this approach, the execution times for depth greater than 6 were larger but did not yield more unit clauses on average. We attempted batch learning on every 500 found clauses using either random or the sorted heuristic. In each batch many of the 500 PR clauses blocked each other because many conflicting PR clauses can be found on a small set of variables in mutilated chess. The PR clauses that were blocked would be found again in following iterations, leading to more reducts generated and solved. This caused much longer execution times. Adding PR clauses instantly is a good configuration for reducing execution time when there are many conflicting clauses. However, for some less symmetric problems it may be worth the tradeoff to learn the clauses in batches, because learning a few bad PR clauses may disrupt the subsequent iterations.

## 7 SAT Competition Benchmarks

We evaluated PRELEARN on previous SAT competition formulas. Formulas from the '13, '15, '16, '19, '20, and '21 SAT competitions' main tracks were grouped by size. **0-10k** contains the 323 formulas with less than 10,000 clauses and **10k-50k** contains

**Table 2.** Fraction of benchmarks where PR clauses were learned, average runtime of PRELEARN, generated positive reducts and satisfiable positive reducts (PR clauses learned), and number of failed literals found.

Set	Benches	Avg. (s)	Generated Reducts	Sat. Reducts	% Sat.	Failed Lits
0-10k	221/323	22.36	104,850,011	548,417	0.52%	3,416
10k-50k	237/348	71.08	163,014,068	789,281	0.48%	6,290

the 348 formulas with between 10,000 and 50,000 clauses. In general, short PR proofs have been found for hard combinatorial problems typically having few clauses (0-10k). These include the pigeonhole and mutilated chessboard problems, some of which appear in 0-10k benchmarks. The PR clauses that can be derived for these formulas are intuitive and almost always beneficial to solvers. Less is known about the impact of PR clauses on larger formulas, motivating our separation of test sets by size. The repository containing the preprocessing tool, experiment configurations, and experiment data can be found at <https://github.com/jreeves3/PReLearn>.

We ran our experiments on StarExec [30]. The specs for the compute nodes can be found online.<sup>1</sup> The compute nodes that ran our experiments were Intel Xeon E5 cores with 2.4 GHz, and all experiments ran with 64 GB of memory and a 5,000 second timeout. We run PRELEARN for 50 iterations over 100 seconds, exiting early if no new PR clauses were found in an iteration.

PRELEARN was executed as a stand-alone program, producing a derivation proof and a modified CNF. For experiments, the CDCL solver KISSAT [5] was called once on the original formula and once on the modified CNF. KISSAT was selected because of its high-rankings in previous SAT competitions, but we expect the results to generalize to other CDCL SAT solvers.

Derivation proofs from PRELEARN were verified in all solved instances using the independent proof checker DPR-TRIM using a forward check. This can be extended to complete proofs in the following way. In the unsatisfiable case the proof for the learned PR clauses is concatenated to the proof traced by KISSAT, and the complete proof is verified against the original formula. In the satisfiable case the partial proof for the learned PR clauses is verified using a forward check in DPR-TRIM, and the satisfying assignment found by KISSAT is verified by the StarExec post-processing tool. Due to resource limitations, we verified a subset of complete proofs in DPR-TRIM. This is more costly because it involves running KISSAT with proof logging, then running DPR-TRIM on the complete proof.

Table 2 shows the cumulative statistics for running PRELEARN on the benchmark sets. Note the number of satisfiable reducts is the number of learned PR clauses, because PR clauses are learned immediately after the reduct is solved. These include both unit and binary PR clauses. A very small percentage of generated reducts is satisfiable, and subsequently learned. This is less important for small formulas when reducts can be computed quickly and there are fewer candidates to consider. However, for the 10k-50k formulas the average runtime more than triples but the number of generated reducts

<sup>1</sup> <https://starexec.org/starexec/public/about.jsp>

**Table 3.** Number of total solved instances and exclusive solved instances running KISSAT with and without PRELEARN. Number of improved instances running KISSAT with PRELEARN. PRELEARN execution times were included in total execution times.

	0-10k SAT	0-10k UNSAT	10k-50k SAT	10k-50k UNSAT
Total w/ PRELEARN	84	149	143	89
Total w/o PRELEARN	80	141	143	91
Exclusively w/ PRELEARN	4	10	4	1
Exclusively w/o PRELEARN	0	2	4	3
Improved w/ PRELEARN	20	44	25	13

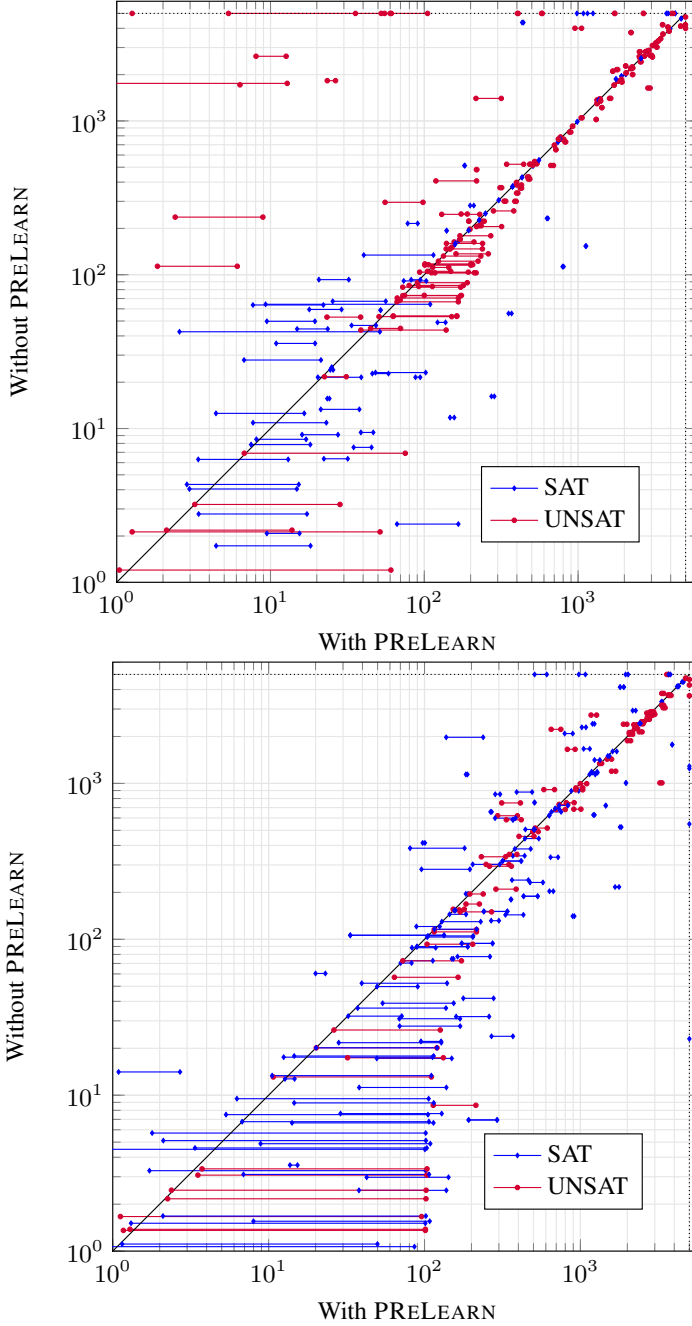
less than doubles. PR clauses are found in about two thirds of the formulas, showing our approach generalizes beyond the canonical problems for which we knew PR clauses existed. Expanding the exploration and increasing the time limit did not help to find PR clauses in the remaining one third.

Table 3 gives a high-level picture of PRELEARN’s impact on KISSAT. PRELEARN significantly improves performance on 0-10k SAT and UNSAT benchmarks. These contain the hard combinatorial problems including pigeonhole that PRELEARN was expected to perform well on. There were 4 additional SAT formulas solved with PRELEARN that KISSAT alone could not solve. This shows that PRELEARN impacts not only hard unsatisfiable problems but satisfiable problems as well. On the other hand, the addition of PR clauses makes some problems more difficult. This is clear with the 10k-50k results, where 5 benchmarks are solved exclusively with PRELEARN and 7 are solved exclusively without. Additionally, PRELEARN improved KISSAT’s performance on 102 of 671 or approx. 15% of benchmarks. This is a large portion of benchmarks, both SAT and UNSAT, for which PRELEARN is helpful.

Figure 4 gives a more detailed picture on the impact of PRELEARN per benchmark. In the scatter plot the left-hand end of each line indicates the KISSAT execution time, while the length of the line indicates the PRELEARN execution time, and so the right-hand end gives the total time for PRELEARN plus KISSAT. Lines that cross the diagonal indicate that the preprocessing improved KISSAT’s performance but ran for longer than the improvement. PRELEARN improved performance for points above the diagonal. Points on the dotted-lines (timeout) are solved by one configuration and not the other.

The top plot gives the results for the 0-10k formulas, with many points on the top timeout line as expected. These are the hard combinatorial problems that can only be solved with PRELEARN. In general, the unsatisfiable formulas benefit more than the satisfiable formulas. PR clauses can reduce the number of solutions in a formula and this may explain the negative impact on many satisfiable formulas. However, there are still some satisfiable formulas that are only solved with PRELEARN.

In the bottom plot, formulas that take a long time to solve (above the diagonal in the upper right-hand corner) are helped more by PRELEARN. In the bottom half of the plot, many lines cross the diagonal meaning the addition of PR clauses provided a negligible benefit. For this set there are more satisfiable formulas for which PRELEARN is helpful.



**Fig. 4.** Execution times w/ and w/o PRELEARN on 0-10k (top) and 10k-50k (bottom) benchmarks. The left-hand point of each segment shows the time for the SAT solver alone; the right-hand point indicates the combined time for preprocessing and solving.

**Table 4.** Some formulas solved by KISSAT exclusively *with* PRELEARN (top) and some formulas solved exclusively *without* PRELEARN (bottom). (\*) solved without KISSAT. Clauses include PR clauses and failed literals learned.

Set	Value	With	Without	Clauses	Formula	Year
0-10k	UNSAT	1.26	–	2,033	ph12*	2013
0-10k	UNSAT	35.69	–	20,179	Pb-chnl15-16.c18*	2019
0-10k	UNSAT	105.01	–	46,759	Pb-chnl20-21.c18	2019
0-10k	UNSAT	59.99	–	1,633	randomG-Mix-n17-d05	2021
0-10k	UNSAT	61.08	–	1,472	randomG-n17-d05	2021
0-10k	UNSAT	407.51	–	1,640	randomG-n18-d05	2021
0-10k	UNSAT	584.95	–	1,706	randomG-Mix-n18-d05	2021
0-10k	SAT	1,082.62	–	9,650	fsf-300-354-2-2-3-2.23.opt	2013
0-10k	SAT	1,250.82	–	10,058	fsf-300-354-2-2-3-2.46.opt	2013
10k-50k	SAT	1,076.34	–	804	sp5-26-19-bin-stri-flat-noid	2021
10k-50k	SAT	608.48	–	901	sp5-26-19-una-nons-tree-noid	2021
10k-50k	SAT	–	22.99	254	Ptn-7824-b13	2016
10k-50k	SAT	–	549.27	133	Ptn-7824-b09	2016
10k-50k	SAT	–	1,246.42	39	Ptn-7824-b02	2016
10k-50k	SAT	–	1,290.49	121	Ptn-7824-b08	2016
10k-50k	UNSAT	–	3,650.21	31,860	rphp4_110_shuffled	2016
10k-50k	UNSAT	–	4,273.88	31,531	rphp4_115_shuffled	2016

The results in Figure 4 are encouraging, with many formulas significantly benefiting from PRELEARN. PRELEARN improves the performance on both SAT and UNSAT formulas of varying size and difficulty. In addition, lines that cross the diagonal imply that improving the runtime efficiency of PRELEARN alone would produce more improved instances. For future work, it would be beneficial to classify formulas before running PRELEARN. There may exist general properties of a formula that signal when PRELEARN will be useful and when PRELEARN will be harmful to a CDCL solver. For instance, a formula’s community structure [2] may help focus the search to parts of the formula where PR clauses are beneficial.

## 7.1 Benchmark Families

In this section we analyze benchmark families that PRELEARN had the greatest positive (negative) effect on, found in Table 4. Studying the formulas PRELEARN works well on may reveal better heuristics for finding good PR clauses.

It has been shown that PR works well for hard combinatorial problems based on perfect matchings [14, 15]. The perfect matching benchmarks (randomG) [7] are a generalization of the pigeonhole (php) and mutilated chessboard problems with varying at-most-one encodings and edge densities. The binary PR clauses can be intuitively understood as blocking two edges from the perfect matching if there exists two other edges that match the same nodes. These benchmarks are relatively small but extremely hard for CDCL solvers. Symmetry-breaking with PR clauses greatly reduces the search space and leads KISSAT to a short proof of unsatisfiability. PRELEARN also benefits

other hard combinatorial problems that use pseudo-Boolean constraints. The pseudo-Boolean (Pb-chnl) [24] benchmarks are based on at-most-one constraints (using the pairwise encoding) and at-least-one constraints. These formulas have a similar graphical structure to the perfect matching benchmarks. Binary PR clauses block two edges when another set of edges exists that are incident to the same nodes.

For the other two benchmark families that benefited from PRELEARN, the intuition behind PR learning is less clear. The fixed-shape random formulas (fsf) [29] are parameterized non-clausal random formulas built from hyper-clauses. The SAT encoding makes use of the Plaisted-Greenbaum transformation, introducing circuit-like structure to the problem. The superpermutation problem (sp) [22] asks whether a sequence of digits  $1-n$  of length  $l$  can contain every permutation of  $[1, n]$  as a subsequence, and the optimization variant asks for the smallest such  $l$  given  $n$ . The sequence of  $l$  digits is encoded directly and passed through a multi-layered circuit that checks for the existence of each individual permutation. Digits use the binary (*bin*) or unary (*una*) encoding, are strict *stri* if clauses constrain digit bits to valid encodings and nonstrict *nons* otherwise, and *flat* if the circuit is a large AND or *tree* for prefix recognizing nested circuits. The formulas given ask to find a prefix of a superpermutation for  $n = 5$  or length 26 with 19 permutations. The check for 19 permutations was encoded as cardinality constraints in a pseudo-Boolean instance, then converted back to SAT. Each individual permutation is checked by duplicating circuits at each possible starting position of the permutation in  $l$ . PR clauses may be pruning certain starting positions for some permutations or affecting the pseudo-Boolean constraints. This cannot be determined without a deeper knowledge of the benchmark generator.

The relativized pigeonhole problem (rphp) [3] involves placing  $k$  pigeons in  $k - 1$  holes with  $n$  nesting places. This problem has polynomial hardness for resolution, unlike the exponential hardness of the classical pigeonhole problem. The symmetry-breaking preprocessor BREAKID [9] generates symmetry-breaking formulas for rphp that are easy for a CDCL solver. PRELEARN can learn many PR clauses but the formula does not become easier. Note PRELEARN can solve the php with  $n = 12$  in a second.

One problem is clause and variable permuting (a.k.a. shuffling). The mutilated chessboard problem can still be solved by PRELEARN after permuting variables and clauses. The pigeonhole problem can be solved after permuting clauses but not after permuting variable names. In PRELEARN, PR candidates are sorted by variable name independent of clause ordering, but when the variable names change the order of learned clauses changes. In the mutilated chessboard problem there is local structure, so similar PR clauses are learned under variable renaming. In the pigeonhole problem there is global structure, so a variable renaming can significantly change the binary PR clauses learned and cause earlier saturation with far fewer units.

Another problem is that the addition of PR clauses can change the existing structure of a formula and negatively affect CDCL heuristics. The Pythagorean Triples Problem (Ptn) [19] asks whether monochromatic solutions of the equation  $a^2 + b^2 = c^2$  can be avoided. The formulas encode numbers  $\{1, \dots, 7824\}$ , for which a valid 2-coloring is possible. In the namings, the  $N$  in  $bN$  denotes the number of backbone literals added to the formula. A backbone literal is a literal assigned true in every solution. Adding more than 20 backbone literals makes the problem easy. For each formula KISSAT can



find a satisfying assignment, but timeouts with the addition of PR clauses. For one instance, adding only 39 PR clauses will lead to a timeout. In some hard SAT and UNSAT problems solvers require some amount of luck and adding a few clauses or shuffling a formula can cause a CDCL solver’s performance to sharply decrease. The Pythagorean Triples problem was originally solved with a local search solver, and local search still performs well after adding PR clauses.

In a straight-forward way, one can avoid the negative effects of adding harmful PR clauses by running two solvers in parallel: one with PRELEARN and one without. This fits with the portfolio approach for solving SAT problems.

## 8 Conclusion and Future Work

In this paper we presented PRELEARN, a tool built from the SADICAL framework that learns PR clauses in a preprocessing stage. We developed several heuristics for finding PR clauses and multiple configurations for clause learning. In the evaluation we found that PRELEARN improves the performance of the CDCL solver KISSAT on many benchmarks from past SAT competitions.

For future work, quantifying the usefulness of each PR clause in relation to guiding the CDCL solver may lead to better learning heuristics. This is a difficult task that likely requires problem specific information. Separately, failed clause caching can improve performance by remembering and avoiding candidate clauses that fail with unsatisfiable reducts in multiple iterations. This would be most beneficial for problems like the mutilated chessboard that have many conflicting PR clauses. Lastly, incorporating PRELEARN during in-processing may allow for more PR clauses to be learned. This could be implemented with the inner/outer solver framework but would require a significantly narrowed search. CDCL learns many clauses during execution and it would be infeasible to examine binary PR clauses across the entire formula.

**Acknowledgements.** We thank the community at StarExec for providing computational resources.

## References

1. Alekhnovich, M.: Mutilated chessboard problem is exponentially hard for resolution. *Theoretical Computer Science* **310**(1), 513–525 (2004)
2. Ansótegui, C., Bonet, M.L., Giráldez-Cru, J., Levy, J., Simon, L.: Community structure in industrial SAT instances. *Journal of Artificial Intelligence Research (JAR)* **66**, 443–472 (2019)
3. Atserias, A., Lauria, M., Nordström, J.: Narrow proofs may be maximally long. *ACM Transactions on Computational Logic* **17**(3) (2016)
4. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning SAT solvers. In: *AAAI Conference on Artificial Intelligence*. pp. 15–20. AAAI Press (2010)
5. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020 (2020), unpublished
6. Biere, A., Fröhlich, A.: Evaluating CDCL variable scoring schemes. In: *Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 9340, pp. 405–422 (2015)
7. Codel, C.R., Reeves, J.E., Heule, M.J.H., Bryant, R.E.: Bipartite perfect matching benchmarks. In: *Pragmatics of SAT* (2021)
8. Cook, S.A.: A short proof of the pigeon hole principle using extended resolution. *SIGACT News* **8**(4), 28–32 (1976)
9. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: *Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 9710, pp. 104–122. Springer (2016)
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: *Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 3569, pp. 61–75. Springer (2005)
11. Freeman, J.W.: Improvements to Propositional Satisfiability Search Algorithms. Ph.D. thesis, USA (1995)
12. Haken, A.: The intractability of resolution. *Theoretical Computer Science* **39**, 297–308 (1985)
13. Heule, M.J.H., Hunt, W.A., Wetzler, N.: Expressing symmetry breaking in DRAT proofs. In: *Conference on Automated Deduction (CADE)*. LNCS, vol. 9195, pp. 591–606. Springer (2015)
14. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: *Conference on Automated Deduction (CADE)*. LNCS, vol. 10395, pp. 130–147. Springer (2017)
15. Heule, M.J.H., Kiesl, B., Biere, A.: Clausal proofs of mutilated chessboards. In: *NASA Formal Methods*. LNCS, vol. 11460, pp. 204–210 (2019)
16. Heule, M.J.H., Kiesl, B., Biere, A.: Encoding redundancy for satisfaction-driven clause learning. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. LNCS, vol. 11427, pp. 41–58. Springer (2019)
17. Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension free proof systems. In: *Journal of Automated Reasoning*. vol. 64, pp. 533–544 (2020)
18. Heule, M.J.H., Kiesl, B., Seidl, M., Biere, A.: PRunning through satisfaction. In: *Haifa Verification Conference (HVC)*. LNCS, vol. 10629, pp. 179–194 (2017)
19. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In: *Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 9710, pp. 228–245. Springer (2016)
20. Heule, M.J., Hunt, W.A., Wetzler, N.: Trimming while checking clausal proofs. In: *Formal Methods in Computer-Aided Design (FMCAD)*. pp. 181–188 (2013)
21. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: *International Joint Conference on Automated Reasoning (IJCAR)*. LNCS, vol. 7364, pp. 355–370. Springer (2012)

22. Johnston, N.: Non-uniqueness of minimal superpermutations. *Discrete Mathematics* **313**(14), 1553–1557 (2013)
23. Kiesel, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: *International Joint Conference on Automated Reasoning (IJCAR)*. LNCS, vol. 9706, pp. 45–61 (2016)
24. Lecoutre, C., Roussel, O.: XCSP3 competition 2018 proceedings. pp. 40–41 (2018)
25. Liang, J., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: *Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 9710, pp. 123–140 (2016)
26. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of Boolean formulas. In: *Haifa Verification Conference (HVC)*. LNCS, vol. 7857, pp. 102–117 (2013)
27. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: *Handbook of Satisfiability*, pp. 131–153. IOS Press (2009)
28. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: *Proceedings of the 38th Annual Design Automation Conference*. p. 530–535. ACM (2001)
29. Navarro, J.A., Voronkov, A.: Generation of hard non-clausal random satisfiability problems. In: *AAAI Conference on Artificial Intelligence*. pp. 436–442. The MIT Press (2005)
30. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: *International Joint Conference on Automated Reasoning (IJCAR)*. LNCS, vol. 8562, pp. 367–373. Springer (2014)
31. Tan, Y.K., Heule, M.J.H., Myreen, M.O.: cake\_lpr: Verified propagation redundancy checking in CakeML. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part II*. LNCS, vol. 12652, pp. 223–241 (2021)
32. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer (1983)
33. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: *Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 8561, pp. 422–429 (2014)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Reasoning About Vectors Using an SMT Theory of Sequences

Ying Sheng<sup>1(✉)</sup>, Andres Nötzli<sup>1</sup>, Andrew Reynolds<sup>2</sup>, Yoni Zohar<sup>3</sup>, David Dill<sup>4</sup>, Wolfgang Grieskamp<sup>4</sup>, Junkil Park<sup>4</sup>, Shaz Qadeer<sup>4</sup>, Clark Barrett<sup>1</sup>, and Cesare Tinelli<sup>2</sup>

<sup>1</sup> Stanford University, Stanford, USA

ying1123@stanford.edu

<sup>2</sup> The University of Iowa, Iowa City, USA

<sup>3</sup> Bar-Ilan University, Ramat Gan, Israel

<sup>4</sup> Meta Novi, Menlo Park, USA

**Abstract.** Dynamic arrays, also referred to as vectors, are fundamental data structures used in many programs. Modeling their semantics efficiently is crucial when reasoning about such programs. The theory of arrays is widely supported but is not ideal, because the number of elements is fixed (determined by its index sort) and cannot be adjusted, which is a problem, given that the length of vectors often plays an important role when reasoning about vector programs. In this paper, we propose reasoning about vectors using a theory of sequences. We introduce the theory, propose a basic calculus adapted from one for the theory of strings, and extend it to efficiently handle common vector operations. We prove that our calculus is sound and show how to construct a model when it terminates with a saturated configuration. Finally, we describe an implementation of the calculus in *cvc5* and demonstrate its efficacy by evaluating it on verification conditions for smart contracts and benchmarks derived from existing array benchmarks.

## 1 Introduction

Generic vectors are used in many programming languages. For example, in C++’s standard library, they are provided by `std::vector`. Automated verification of software systems that manipulate vectors requires an efficient and automated way of reasoning about them. Desirable characteristics of any approach for reasoning about vectors include: (i) expressiveness—operations that are commonly performed on vectors should be supported; (ii) generality—vectors are always “vectors of” some type (e.g., vectors of integers), and so it is desirable that vector reasoning be integrated within a more general framework; solvers for satisfiability modulo theories (SMT) provide such a framework and are widely used in verification tools (see [5] for a recent survey); (iii) efficiency—fast and efficient reasoning is essential for usability, especially as verification tools are increasingly used by non-experts and in continuous integration.

This work was funded in part by the Stanford Center for Blockchain Research, NSF-BSF grant numbers 2110397 (NSF) and 2020704 (BSF), and Meta Novi. Part of the work was done when the first author was an intern at Meta Novi.

© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 125–143, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_9](https://doi.org/10.1007/978-3-031-10769-6_9)

Despite the ubiquity of vectors in software on the one hand and the effectiveness of SMT solvers for software verification on the other hand, there is not currently a clean way to represent vectors using operators from the SMT-LIB standard [3]. While the theory of arrays can be used, it is not a great fit because arrays have a fixed size determined by their index type. Representing a dynamic array thus requires additional modeling work. Moreover, to reach an acceptable level of expressivity, quantifiers are needed, which often makes the reasoning engine less efficient and robust. Indeed, part of the motivation for this work was frustration with array-based modeling in the Move Prover, a verification framework for smart contracts [24] (see Sect. 6 for more information about the Move Prover and its use of vectors). The current paper bridges this gap by studying and implementing a native theory of *sequences* in the SMT framework, which satisfies the desirable properties for vector reasoning listed above.

We present two SMT-based calculi for determining satisfiability in the theory of sequences. Since the decidability of even weaker theories is unknown (see, e.g., [9, 15]), we do not aim for a decision procedure. Rather, we prove model and solution soundness (that is, when our procedure terminates, the answer is correct). Our first calculus leverages techniques for the theory of strings. We generalize these techniques, lifting rules specific to string characters to more general rules for arbitrary element types. By itself, this base calculus is already quite effective. However, it misses opportunities to perform high-level vector-based reasoning. For example, both reading from and updating a vector are very common operations in programming, and reasoning efficiently about the corresponding sequence operators is thus crucial. Our second calculus addresses this gap by integrating reasoning methods from array solvers (which handle reads and updates efficiently) into the first procedure. Notice, however, that this integration is not trivial, as it must handle novel combinations of operators (such as the combination of update and read operators with concatenation) as well as out-of-bounds cases that do not occur with ordinary arrays. We have implemented both variants of our calculus in the *cvc5* SMT solver [2] and evaluated them on benchmarks originating from the Move prover, as well as benchmarks that were translated from SMT-LIB array benchmarks.

As is typical, both of our calculi are agnostic to the sort of the elements in the sequence. Reasoning about sequences of elements from a particular theory can then be done via theory combination methods such as Nelson-Oppen [18] or polite combination [16, 20]. The former can be done for stably infinite theories (and the theory of sequences that we present here is stably infinite), while the latter requires investigating the politeness of the theory, which we expect to do in future work.

The rest of the paper is organized as follows. Section 2 includes basic notions from first-order logic. Section 3 introduces the theory of sequences and shows how it can be used to model vectors. Section 4 presents calculi for this theory and discusses their correctness. Section 5 describes the implementation of these calculi in *cvc5*. Section 6 presents an evaluation comparing several variations of the sequence solver in *cvc5* and Z3. We conclude in Sect. 7 with directions for further research.

**Related Work:** Our work crucially builds on a proposal by Bjørner et al. [8], but extends it in several key ways. First, their implementation (for a logic they call *QF\_BVRE*) restricts the generality of the theory by allowing only bit-vector elements (representing characters) and assuming that sequences are bounded. In contrast, our

calculus maintains full generality, allowing unbounded sequences and elements of arbitrary types. Second, while our core calculus focuses only on a subset of the operators in [8], our implementation supports the remaining operators by reducing them to the core operators, and also adds native support for the update operator, which is not included in [8].

The base calculus that we present for sequences builds on similar work for the theory of strings [6, 17]. We extend our base calculus to support array-like reasoning based on the weak-equivalence approach [10]. Though there exists some prior work on extending the theory of arrays with more operators and reasoning about length [1, 12, 14], this work does not include support for most of the of the sequence operators we consider here.

The SMT-solver Z3 [11] also provides a solver for sequences. However, its documentation is limited [7], it does not support update directly, and its internal algorithms are not described in the literature. Furthermore, as we show in Sect. 6, the performance of the Z3 implementation is generally inferior to our implementation in *cvc5*.

## 2 Preliminaries

We assume the usual notions and terminology of many-sorted first-order logic with equality (see, e.g., [13] for a complete presentation). We consider many-sorted signatures  $\Sigma$ , each containing a set of sort symbols (including a Boolean sort *Bool*), a family of logical symbols  $\approx$  for equality, with sort  $\sigma \times \sigma \rightarrow \text{Bool}$  for all sorts  $\sigma$  in  $\Sigma$  and interpreted as the identity relation, and a set of interpreted (and sorted) function symbols. We assume the usual definitions of well-sorted terms, literals, and formulas as terms of sort *Bool*. A literal is *flat* if it has the form  $\perp$ ,  $p(x_1, \dots, x_n)$ ,  $\neg p(x_1, \dots, x_n)$ ,  $x \approx y$ ,  $\neg x \approx y$ , or  $x \approx f(x_1, \dots, x_n)$ , where  $p$  and  $f$  are function symbols and  $x$ ,  $y$ , and  $x_1, \dots, x_n$  are variables. A  $\Sigma$ -interpretation  $\mathcal{M}$  is defined as usual, satisfying  $\mathcal{M}(\perp) = \text{false}$  and assigns: a set  $\mathcal{M}(\sigma)$  to every sort  $\sigma$  of  $\Sigma$ , a function  $\mathcal{M}(f) : \mathcal{M}(\sigma_1) \times \dots \times \mathcal{M}(\sigma_n) \rightarrow \mathcal{M}(\sigma)$  to any function symbol  $f$  of  $\Sigma$  with arity  $\sigma_1 \times \dots \times \sigma_n \rightarrow \sigma$ , and an element  $\mathcal{M}(x) \in \mathcal{M}(\sigma)$  to any variable  $x$  of sort  $\sigma$ . The satisfaction relation between interpretations and formulas is defined as usual and is denoted by  $\models$ .

A *theory* is a pair  $T = (\Sigma, \mathbf{I})$ , in which  $\Sigma$  is a signature and  $\mathbf{I}$  is a class of  $\Sigma$ -interpretations, closed under variable reassignment. The *models* of  $T$  are the interpretations in  $\mathbf{I}$  without any variable assignments. A  $\Sigma$ -formula  $\varphi$  is *satisfiable* (resp., *unsatisfiable*) in  $T$  if it is satisfied by some (resp., no) interpretation in  $\mathbf{I}$ . Given a (set of) terms  $S$ , we write  $\mathcal{T}(S)$  to denote the set of all subterms of  $S$ . For a theory  $T = (\Sigma, \mathbf{I})$ , a set  $S$  of  $\Sigma$ -formulas and a  $\Sigma$ -formula  $\varphi$ , we write  $S \models_T \varphi$  if every interpretation  $\mathcal{M} \in \mathbf{I}$  that satisfies  $S$  also satisfies  $\varphi$ . By convention and unless otherwise stated, we use letters  $w, x, y, z$  to denote variables and  $s, t, u, v$  to denote terms.

The theory  $T_{\text{LIA}} = (\Sigma_{\text{LIA}}, \mathbf{I}_{T_{\text{LIA}}})$  of *linear integer arithmetic* is based on the signature  $\Sigma_{\text{LIA}}$  that includes a single sort *Int*, all natural numbers as constant symbols, the unary  $-$  symbol, the binary  $+$  symbol and the binary  $\leq$  relation. When  $k \in \mathbb{N}$ , we use the notation  $k \cdot x$ , inductively defined by  $0 \cdot x = 0$  and  $(m + 1) \cdot x = x + m \cdot x$ . In turn,  $\mathbf{I}_{T_{\text{LIA}}}$  consists of all structures  $\mathcal{M}$  for  $\Sigma_{\text{LIA}}$  in which the domain  $\mathcal{M}(\text{Int})$  of *Int* is the set

Symbol	Arity	SMT-LIB	Description
$n$	Int	$n$	All constants $n \in \mathbb{N}$
$+$	$\text{Int} \times \text{Int} \rightarrow \text{Int}$	$+$	Integer addition
$-$	$\text{Int} \rightarrow \text{Int}$	$-$	Unary Integer minus
$\leq$	$\text{Int} \times \text{Int} \rightarrow \text{Bool}$	$<=$	Integer inequality
$\epsilon$	Seq	<code>seq.empty</code>	The empty sequence
<code>unit</code>	$\text{Elem} \rightarrow \text{Seq}$	<code>seq.unit</code>	Sequence constructor
<code> _</code>	$\text{Seq} \rightarrow \text{Int}$	<code>seq.len</code>	Sequence length
<code>nth</code>	$\text{Seq} \times \text{Int} \rightarrow \text{Elem}$	<code>seq.nth</code>	Element access
<code>update</code>	$\text{Seq} \times \text{Int} \times \text{Elem} \rightarrow \text{Seq}$	<code>seq.update</code>	Element update
<code>extract</code>	$\text{Seq} \times \text{Int} \times \text{Int} \rightarrow \text{Seq}$	<code>seq.extract</code>	Extraction (subsequence)
<code>_ ++ \cdots ++ _</code>	$\text{Seq} \times \cdots \times \text{Seq} \rightarrow \text{Seq}$	<code>seq.concat</code>	Concatenation

**Fig. 1.** Signature for the theory of sequences.

of integer numbers, for every constant symbol  $n \in \mathbb{N}$ ,  $\mathcal{M}(n) = n$ , and  $+$ ,  $-$ , and  $\leq$  are interpreted as usual. We use standard notation for integer intervals (e.g.,  $[a, b]$  for the set of integers  $i$ , where  $a \leq i \leq b$  and  $[a, b)$  for the set where  $a \leq i < b$ ).

### 3 A Theory of Sequences

We define the theory  $T_{\text{Seq}}$  of sequences. Its signature  $\Sigma_{\text{Seq}}$  is given in Fig. 1. It includes the sorts Seq, Elem, Int, and Bool, intuitively denoting sequences, elements, integers, and Booleans, respectively. The first four lines include symbols of  $\Sigma_{\text{LIA}}$ . We write  $t_1 \bowtie t_2$ , with  $\bowtie \in \{>, <, \leq\}$ , as syntactic sugar for the equivalent literal expressed using  $\leq$  (and possibly  $\neg$ ). The sequence symbols are given on the remaining lines. Their arities are also given in Fig. 1. Notice that `_ ++  $\cdots$  ++ _` is a variadic function symbol.

Interpretations  $\mathcal{M}$  of  $T_{\text{Seq}}$  interpret: Int as the set of integers; Elem as some set; Seq as the set of finite sequences whose elements are from Elem;  $\epsilon$  as the empty sequence; `unit` as a function that takes an element from  $\mathcal{M}(\text{Elem})$  and returns the sequence that contains only that element; `nth` as a function that takes an element  $s$  from  $\mathcal{M}(\text{Seq})$  and an integer  $i$  and returns the  $i$ th element of  $s$ , in case  $i$  is non-negative and is smaller than the length of  $s$  (we take the first element of a sequence to have index 0). Otherwise, the function has no restrictions; `update` as a function that takes an element  $s$  from  $\mathcal{M}(\text{Seq})$ , an integer  $i$ , and an element  $a$  from  $\mathcal{M}(\text{Elem})$  and returns the sequence obtained from  $s$  by replacing its  $i$ th element by  $a$ , in case  $i$  is non-negative and smaller than the length of  $s$ . Otherwise, the returned value is  $s$  itself; `extract` as a function that takes a sequence  $s$  and integers  $i$  and  $j$ , and returns the maximal sub-sequence of  $s$  that starts at index  $i$  and has length at most  $j$ , in case both  $i$  and  $j$  are non-negative and  $i$  is smaller than the length of  $s$ . Otherwise, the returned value is the empty sequence;<sup>1</sup> `|_` as a function that takes a sequence and returns its length; and `_ ++  $\cdots$  ++ _` as a function that takes some number of sequences (at least 2) and returns their concatenation.

<sup>1</sup> In [8], the second argument  $j$  denotes the end index, while here it denotes the length of the sub-sequence, in order to be consistent with the theory of strings in the SMT-LIB standard.

Notice that the interpretations of `Elem` and `nth` are not completely fixed by the theory: `Elem` can be set arbitrarily, and `nth` is only defined by the theory for some values of its second argument. For the rest, it can be set arbitrarily.

### 3.1 Vectors as Sequences

We show the applicability of  $T_{\text{Seq}}$  by using it for a simple verification task. Consider the C++ function `swap` at the top of Fig. 2. This function swaps two elements in a vector. The comments above the function include a partial specification for it: if both indexes are in-bounds and the indexed elements are equal, then the function should not change the vector (this is expressed by `s_out==s`). We now consider how to encode the verification condition induced by the code and the specification. The function variables  $a$ ,  $b$ ,  $i$ , and  $j$  can be encoded as variables of sort `Int` with the same names. We include two copies of  $s$ :  $s$  for its value at the beginning, and  $s_{\text{out}}$  for its value at the end. But what should be the sorts of  $s$  and  $s_{\text{out}}$ ? In Fig. 2 we consider two options: one is based on arrays and the other on sequences.

*Example 1 (Arrays).* The theory of arrays includes three sorts: index, element (in this case, both are `Int`), and an array sort `Arr`, as well as two operators:  $x[i]$ , interpreted as the  $i$ th element of  $x$ ; and  $x[i \leftarrow a]$ , interpreted as the array obtained from  $x$  by setting the element at index  $i$  to  $a$ . We declare  $s$  and  $s_{\text{out}}$  as variables of an uninterpreted sort  $V$  and declare two functions  $\ell$  and  $c$ , which, given  $v$  of sort  $V$ , return its length (of sort `Int`) and content (of sort `Arr`), respectively.<sup>2</sup>

Next, we introduce functions to model vector operations:  $\approx_{\mathbf{A}}$  for comparing vectors,  $\text{nth}_{\mathbf{A}}$  for reading from them, and  $\text{update}_{\mathbf{A}}$  for updating them. These functions need to be axiomatized. We include two axioms (bottom of Fig. 2):  $Ax_1$  states that two vectors are equal iff they have the same length and the same contents.  $Ax_2$  axiomatizes the update operator; the result has the same length, and if the updated index is in bounds, then the corresponding element is updated. These axioms are not meant to be complete, but are rather just strong enough for the example.

The first two lines of the `swap` function are encoded as equalities using  $\text{nth}_{\mathbf{A}}$ , and the last two lines are combined into one nested constraint that involves  $\text{update}_{\mathbf{A}}$ . The precondition of the specification is naturally modeled using  $\text{nth}_{\mathbf{A}}$ , and the post-condition is negated, so that the unsatisfiability of the formula entails the correctness of the function w.r.t. the specification. Indeed, the conjunction of all formulas in this encoding is unsatisfiable in the combined theories of arrays, integers, and uninterpreted functions.

The above encoding has two main shortcomings: It introduces auxiliary symbols, and it uses quantifiers, thus reducing clarity and efficiency. In the next example, we see how using the theory of sequences allows for a much more natural and succinct encoding.

*Example 2 (Sequences).* In the sequences encoding,  $s$  and  $s_{\text{out}}$  have sort `Seq`. No auxiliary sorts or functions are needed, as the theory symbols can be used directly. Further,

<sup>2</sup> It is possible to obtain a similar encoding using the theory of datatypes; however, here we use uninterpreted functions which are simpler and better supported by SMT solvers.



```

// @pre: 0 <= i, j < s.size() and s[i] == s[j]
// @post: s.out == s
void swap(std::vector<int>& s, int i, int j) {
    int a = s[i];
    int b = s[j];
    s[i] = b;
    s[j] = a;
}

```

	Sequences	Arrays
Problem Variables	$a, b, i, j : \text{Int} \quad s, s_{out} : \text{Seq}$	$a, b, i, j : \text{Int} \quad c : V \rightarrow \text{Arr} \quad s, s_{out} : V$
Auxiliary Variables		$\ell : V \rightarrow \text{Int} \quad c : V \rightarrow \text{Arr}$ $\approx_{\mathbf{A}} : V \times V \rightarrow \text{Bool}$ $\text{nth}_{\mathbf{A}} : V \times \text{Int} \rightarrow \text{Int}$ $\text{update}_{\mathbf{A}} : V \times \text{Int} \times \text{Int} \rightarrow V$
Axioms		$Ax_1 \wedge Ax_2$
Program	$a \approx \text{nth}(s, i) \wedge b \approx \text{nth}(s, j)$ $s_{out} \approx \text{update}(\text{update}(s, i, b), j, a)$	$a \approx \text{nth}_{\mathbf{A}}(s, i) \wedge b \approx \text{nth}_{\mathbf{A}}(s, j)$ $s_{out} \approx_{\mathbf{A}} \text{update}_{\mathbf{A}}(\text{update}_{\mathbf{A}}(s, i, b), j, a)$
Spec.	$0 \leq i, j <  s  \wedge \text{nth}(s, i) \approx \text{nth}(s, j)$ $\neg s_{out} \approx s$	$0 \leq i, j < \ell(s) \wedge \text{nth}_{\mathbf{A}}(s, i) \approx \text{nth}_{\mathbf{A}}(s, j)$ $\neg s_{out} \approx_{\mathbf{A}} s$

$$Ax_1 := \forall x, y. x \approx_{\mathbf{A}} y \leftrightarrow (\ell(x) \approx \ell(y) \wedge \forall 0 \leq i < \ell(x). c(x)[i] \approx c(y)[i])$$

$$Ax_2 := \forall x, y, i, a. y \approx_{\mathbf{A}} \text{update}_{\mathbf{A}}(x, i, a) \rightarrow (\ell(x) \approx \ell(y) \wedge (0 \leq i < \ell(x) \rightarrow c(y) \approx c(x)[i \leftarrow a]))$$

Fig. 2. An example using  $T_{\text{Seq}}$ .

these symbols do not need to be axiomatized as their semantics is fixed by the theory. The resulting formula, much shorter than in Example 2 and with no quantifiers, is unsatisfiable in  $T_{\text{Seq}}$ .

## 4 Calculi

After introducing some definitions and assumptions, we describe a basic calculus for the theory of sequences, which adapts techniques from previous procedures for the theory of strings. In particular, the basic calculus reduces the operators `nth` and `update` by introducing concatenation terms. We then show how to extend the basic calculus by introducing additional rules inspired by solvers for the theory of arrays; the modified calculus can often reason about `nth` and `update` terms directly, avoiding the introduction of concatenation terms (which are typically expensive to reason about).

Given a vector of sequence terms  $\bar{t} = (t_1, \dots, t_n)$ , we use  $\bar{t}$  to denote the term corresponding to the concatenation of  $t_1, \dots, t_n$ . If  $n = 0$ ,  $\bar{t}$  denotes  $\epsilon$ , and if  $n = 1$ ,  $\bar{t}$  denotes  $t_1$ ; otherwise (when  $n > 1$ ),  $\bar{t}$  denotes a concatenation term having  $n$  children. In our calculi, we distinguish between sequence and arithmetic constraints.

**Definition 1.** A  $\Sigma_{\text{Seq}}$ -formula  $\varphi$  is a sequence constraint if it has the form  $s \approx t$  or  $s \not\approx t$ ; it is an arithmetic constraint if it has the form  $s \approx t$ ,  $s \geq t$ ,  $s \not\approx t$ , or  $s < t$  where  $s, t$  are terms of sort `Int`, or if it is a disjunction  $c_1 \vee c_2$  of two arithmetic constraints.

Notice that sequence constraints do not have to contain sequence terms (e.g.,  $x \approx y$  where  $x, y$  are Elem-variables). Also, equalities and disequalities between terms of sort `Int` are both sequence and arithmetic constraints. In this paper we focus on sequence

$$\begin{array}{ll}
|\epsilon| \rightarrow 0 & |\text{unit}(t)| \rightarrow 1 \\
|\text{update}(s, i, t)| \rightarrow |s| & |s_1 ++ \dots ++ s_n| \rightarrow |s_1| + \dots + |s_n| \\
\bar{u} ++ \epsilon ++ \bar{v} \rightarrow \bar{u} ++ \bar{v} & \bar{u} ++ (s_1 ++ \dots ++ s_n) ++ \bar{v} \rightarrow \bar{u} ++ s_1 ++ \dots ++ s_n ++ \bar{v}
\end{array}$$

**Fig. 3.** Rewrite rules for the reduced form  $t \downarrow$  of a term  $t$ , obtained from  $t$  by applying these rules to completion.

constraints and arithmetic constraints. This is justified by the following lemma. (Proofs of this lemma and later results can be found in an extended version of this paper [23].)

**Lemma 1.** *For every quantifier-free  $\Sigma_{\text{Seq}}$ -formula  $\varphi$ , there are sets  $S_1, \dots, S_n$  of sequence constraints and sets  $A_1, \dots, A_n$  of arithmetic constraints such that  $\varphi$  is  $T_{\text{Seq}}$ -satisfiable iff  $S_i \cup A_i$  is  $T_{\text{Seq}}$ -satisfiable for some  $i \in [1, n]$ .*

Throughout the presentation of the calculi, we will make a few simplifying assumptions.

**Assumption 1.** *Whenever we refer to a set  $S$  of sequence constraints, we assume:*

1. *for every non-variable term  $t \in \mathcal{T}(S)$ , there exists a variable  $x$  such that  $x \approx t \in S$ ;*
2. *for every Seq-variable  $x$ , there exists a variable  $\ell_x$  such that  $\ell_x \approx |x| \in S$ ;*
3. *all literals in  $S$  are flat.*

*Whenever we refer to a set of arithmetic constraints, we assume all its literals are flat.*

These assumptions are without loss of generality as any set can easily be transformed into an equisatisfiable set satisfying the assumptions by the addition of fresh variables and equalities. Note that some rules below introduce non-flat literals. In such cases, we assume that similar transformations are done immediately after applying the rule to maintain the invariant that all literals in  $S \cup A$  are flat. Rules may also introduce fresh variables  $k$  of sort Seq. We further assume that in such cases, a corresponding constraint  $\ell_k \approx |k|$  is added to  $S$  with a fresh variable  $\ell_k$ .

**Definition 2.** *Let  $C$  be a set of constraints. We write  $C \models \varphi$  to denote that  $C$  entails formula  $\varphi$  in the empty theory, and write  $\equiv_C$  to denote the binary relation over  $\mathcal{T}(C)$  such that  $s \equiv_C t$  iff  $C \models s \approx t$ .*

**Lemma 2.** *For all set  $S$  of sequence constraints,  $\equiv_S$  is an equivalence relation; furthermore, every equivalence class of  $\equiv_S$  contains at least one variable.*

We denote the equivalence class of a term  $s$  according to  $\equiv_S$  by  $[s]_{\equiv_S}$  and drop the  $\equiv_S$  subscript when it is clear from the context.

In the presentation of the calculus, it will often be useful to normalize terms to what will be called a *reduced form*.

**Definition 3.** *Let  $t$  be a  $\Sigma_{\text{Seq}}$ -term. The reduced form of  $t$ , denoted by  $t \downarrow$ , is the term obtained by applying the rewrite rules listed in Fig. 3 to completion.*

Observe that  $t \downarrow$  is well defined because the given rewrite rules form a terminating rewrite system. This can be seen by noting that each rule reduces the number of applications of sequence operators in the left-hand side term or keeps that number the same but reduces the size of the term. It is not difficult to show that  $\models_{T_{\text{Seq}}} t \approx t \downarrow$ .

We now introduce some basic definitions related to concatenation terms.

**Definition 4.** A concatenation term is a term of the form  $s_1 ++ \dots ++ s_n$  with  $n \geq 2$ . If each  $s_i$  is a variable, it is a variable concatenation term. For a set  $S$  of sequence constraints, a variable concatenation term  $x_1 ++ \dots ++ x_n$  is singular in  $S$  if  $S \not\models x_i \approx \epsilon$  for at most one variable  $x_i$  with  $i \in [1, n]$ . A sequence variable  $x$  is atomic in  $S$  if  $S \not\models x \approx \epsilon$  and for all variable concatenation terms  $s \in T(S)$  such that  $S \models x \approx s$ ,  $s$  is singular in  $S$ .

We lift the concept of atomic variables to atomic representatives of equivalence classes.

**Definition 5.** Let  $S$  be a set of sequence constraints. Assume a choice function  $\alpha : T(S)/\equiv_S \rightarrow T(S)$  that chooses a variable from each equivalence class of  $\equiv_S$ . A sequence variable  $x$  is an atomic representative in  $S$  if it is atomic in  $S$  and  $x = \alpha([x]_{\equiv_S})$ .

Finally, we introduce a relation that is the foundation for reasoning about concatenations.

**Definition 6.** Let  $S$  be a set of sequence constraints. We inductively define a relation  $S \models_{++} x \approx s$ , where  $x$  is a sequence variable in  $S$  and  $s$  is a sequence term whose variables are in  $T(S)$ , as follows:

1.  $S \models_{++} x \approx x$  for all sequence variables  $x \in T(S)$ .
2.  $S \models_{++} x \approx t$  for all sequence variables  $x \in T(S)$  and variable concatenation terms  $t$ , where  $x \approx t \in S$ .
3. If  $S \models_{++} x \approx (\bar{w} ++ y ++ \bar{z})\downarrow$  and  $S \models y \approx t$  and  $t$  is  $\epsilon$  or a variable concatenation term in  $S$  that is not singular in  $S$ , then  $S \models_{++} x \approx (\bar{w} ++ t ++ \bar{z})\downarrow$ .

Let  $\alpha$  be a choice function for  $S$  as defined in Definition 5. We additionally define the entailment relation  $S \models_{++}^* x \approx \bar{y}$ , where  $\bar{y}$  is of length  $n \geq 0$ , to hold if each element of  $\bar{y}$  is an atomic representative in  $S$  and there exists  $\bar{z}$  of length  $n$  such that  $S \models_{++} x \approx \bar{z}$  and  $S \models y_i \approx z_i$  for  $i \in [1, n]$ .

In other words,  $S \models_{++}^* x \approx t$  holds when  $t$  is a concatenation of atomic representatives and is entailed to be equal to  $x$  by  $S$ . In practice,  $t$  is determined by recursively expanding concatenations using equalities in  $S$  until a fixpoint is reached.

*Example 3.* Suppose  $S = \{x \approx y ++ z, y \approx w ++ u, u \approx v\}$  (we omit the additional constraints required by Assumption 1, part 2 for brevity). It is easy to see that  $u, v, w$ , and  $z$  are atomic in  $S$ , but  $x$  and  $y$  are not. Furthermore,  $w$  and  $z$  (and one of  $u$  or  $v$ ) must also be atomic representatives. Clearly,  $S \models_{++} x \approx x$  and  $S \models x \approx y ++ z$ . Moreover,  $y ++ z$  is a variable concatenation term that is not singular in  $S$ . Hence, we have  $S \models_{++} x \approx (y ++ z)\downarrow$ , and so  $S \models_{++} x \approx y ++ z$  (by using either Item 2 or Item 3 of Definition 6, as in fact  $x \approx y ++ z \in S$ ). Now, since  $S \models_{++} x \approx y ++ z$ ,  $S \models y \approx w ++ u$ , and  $w ++ u$  is a variable concatenation term not singular in  $S$ , we get that  $S \models_{++} x \approx ((w ++ u) ++ z)\downarrow$ , and so  $S \models_{++} x \approx w ++ u ++ z$ . Now, assume that  $v = \alpha([v]_{\equiv_S}) = \alpha(\{v, u\})$ . Then,  $S \models_{++}^* x \approx w ++ v ++ z$ .

Our calculi can be understood as modeling abstractly a cooperation between an *arithmetic subsolver* and a *sequence subsolver*. Many of the derivation rules lift those in the string calculus of Liang et al. [17] to sequences of elements of an arbitrary type. We describe them similarly as rules that modify *configurations*.

**Definition 7.** A configuration is either the distinguished configuration *unsat* or a pair  $(S, A)$  of a set  $S$  of sequence constraints and a set  $A$  of arithmetic constraints.

The rules are given in *guarded assignment form*, where the rule premises describe the conditions on the current configuration under which the rule can be applied, and the conclusion is either *unsat*, or otherwise describes the resulting modifications to the configuration. A rule may have multiple conclusions separated by  $\parallel$ . In the rules, some of the premises have the form  $S \models s \approx t$  (see Definition 2). Such entailments can be checked with standard algorithms for congruence closure. Similarly, premises of the form  $S \models_{LIA} s \approx t$  can be checked by solvers for linear integer arithmetic.

An application of a rule is *redundant* if it has a conclusion where each component in the derived configuration is a subset of the corresponding component in the premise configuration. We assume that for rules that introduce fresh variables, the introduced variables are identical whenever the premises triggering the rule are the same (i.e., we cannot generate an infinite sequence of rule applications by continuously using the same premises to introduce fresh variables).<sup>3</sup> A configuration other than *unsat* is *saturated* with respect to a set  $R$  of derivation rules if every possible application of a rule in  $R$  to it is redundant. A *derivation tree* is a tree where each node is a configuration whose children, if any, are obtained by a non-redundant application of a rule of the calculus. A derivation tree is *closed* if all of its leaves are *unsat*. As we show later, a closed derivation tree with root node  $(S, A)$  is a proof that  $A \cup S$  is unsatisfiable in  $T_{Seq}$ . In contrast, a derivation tree with root node  $(S, A)$  and a saturated leaf with respect to all the rules of the calculus is a witness that  $A \cup S$  is satisfiable in  $T_{Seq}$ .

#### 4.1 Basic Calculus

**Definition 8.** The calculus BASE consists of the derivation rules in Figs. 4 and 5.

Some of the rules are adapted from previous work on string solvers [17, 22]. Compared to that work, our presentation of the rules is noticeably simpler, due to our use of the relation  $\models_{++}^*$  from Definition 6. In particular, our configurations consist only of pairs of sets of formulas, without any auxiliary data-structures.

Note that judgments of the form  $S \models_{++}^* x \approx t$  are used in premises of the calculus. It is possible to compute whether such a premise holds thanks to the following lemma.

**Lemma 3.** Let  $S$  be a set of sequence constraints and  $A$  a set of arithmetic constraints. If  $(S, A)$  is saturated w.r.t. *S-Prop*, *L-Intro* and *L-Valid*, the problem of determining whether  $S \models_{++}^* x \approx s$  for given  $x$  and  $s$  is decidable.

Lemma 3 assumes saturation with respect to certain rules. Accordingly, our proof strategy, described in Sect. 5, will ensure such saturation before attempting to apply rules relying on  $\models_{++}^*$ . The relation  $\models_{++}^*$  induces a normal form for each equivalence class of  $\equiv_S$ .

<sup>3</sup> In practice, this is implemented by associating each introduced variable with a *witness term* as described in [21].

$$\begin{array}{c}
\text{A-Conf} \frac{A \models_{\text{LIA}} \perp}{\text{unsat}} \quad \text{A-Prop} \frac{A \models_{\text{LIA}} s \approx t \quad s, t \in \mathcal{T}(S)}{S := S, s \approx t} \\
\\
\text{S-Conf} \frac{S \models \perp}{\text{unsat}} \quad \text{S-Prop} \frac{S \models s \approx t \quad s, t \in \mathcal{T}(S) \quad s, t \text{ are } \Sigma_{\text{LIA}}\text{-terms}}{A := A, s \approx t} \\
\\
\text{S-A} \frac{x, y \in \mathcal{T}(S) \cap \mathcal{T}(A) \quad x, y : \text{Int}}{A := A, x \approx y \quad \parallel \quad A := A, x \not\approx y} \\
\\
\text{L-Intro} \frac{s \in \mathcal{T}(S) \quad s : \text{Seq}}{S := S, |s| \approx (|s|)\downarrow} \quad \text{L-Valid} \frac{x \in \mathcal{T}(S) \quad x : \text{Seq}}{S := S, x \approx \epsilon \quad \parallel \quad A := A, \ell_x > 0} \\
\\
\text{U-Eq} \frac{S \models \text{unit}(x) \approx \text{unit}(y)}{S := S, x \approx y} \quad \text{C-Eq} \frac{S \models_{++}^* x \approx \bar{z} \quad S \models_{++}^* y \approx \bar{z}}{S := S, x \approx y} \\
\\
\text{C-Split} \frac{S \models_{++}^* x \approx (\bar{w} ++ y ++ \bar{z})\downarrow \quad S \models_{++}^* x \approx (\bar{w} ++ y' ++ \bar{z}')\downarrow}{\begin{array}{c} A := A, \ell_y > \ell_{y'} \quad S := S, y \approx y' ++ k \quad \parallel \\ A := A, \ell_y < \ell_{y'} \quad S := S, y' \approx y ++ k \quad \parallel \\ A := A, \ell_y \approx \ell_{y'} \quad S := S, y \approx y' \end{array}} \\
\\
\text{Deq-Ext} \frac{x \not\approx y \in S \quad x, y : \text{Seq}}{A := A, \ell_x \not\approx \ell_y \quad \parallel \quad A := A, \ell_x \approx \ell_y, 0 \leq i < \ell_x \quad S := S, w_1 \approx \text{nth}(x, i), w_2 \approx \text{nth}(y, i), w_1 \not\approx w_2}
\end{array}$$

**Fig. 4.** Core derivation rules. The rules use  $k$  and  $i$  to denote fresh variables of sequence and integer sort, respectively, and  $w_1$  and  $w_2$  for fresh element variables.

**Lemma 4.** *Let  $S$  be a set of sequence constraints and  $A$  a set of arithmetic constraints. Suppose  $(S, A)$  is saturated w.r.t. A-Conf, S-Prop, L-Intro, L-Valid, and C-Split. Then, for every equivalence class  $e$  of  $\equiv_S$  whose terms are of sort Seq, there exists a unique (possibly empty)  $\bar{s}$  such that whenever  $S \models_{++}^* x \approx s'$  for  $x \in e$ , then  $s' = \bar{s}$ . In this case, we call  $\bar{s}$  the normal form of  $e$  (and of  $x$ ).*

We now turn to the description of the rules in Fig. 4, which form the core of the calculus. For greater clarity, some of the conclusions of the rules include terms before they are flattened. First, either subsolver can report that the current set of constraints is unsatisfiable by using the rules A-Conf or S-Conf. For the former, the entailment  $\models_{\text{LIA}}$  (which abbreviates  $\models_{T_{\text{LIA}}}$ ) can be checked by a standard procedure for linear integer arithmetic, and the latter corresponds to a situation where congruence closure detects a conflict between an equality and a disequality. The rules A-Prop, S-Prop, and S-A correspond to a form of Nelson-Oppen-style theory combination between the two subsolvers. The first two communicate equalities between the sub-solvers, while the third guesses arrangements for shared variables of sort Int. L-Intro ensures that the length term  $|s|$  for each sequence term  $s$  is equal to its reduced form  $(|s|)\downarrow$ . L-Valid restricts sequence lengths to be non-negative, splitting on whether each sequence is empty or has a length greater than 0. The unit operator is injective, which is captured by U-Eq. C-Eq concludes that two sequence terms are equal if they have the same normal form. If two sequence variables have different normal forms, then C-Split takes the first differing components  $y$  and  $y'$  from the two normal forms and splits on their length relationship. Note that C-Split is the source for non-termination of the calculus (see, e.g., [17, 22]).

$$\begin{array}{c}
\text{R-Extract} \frac{x \approx \text{extract}(y, i, j) \in S}{\begin{array}{l} A := A, i < 0 \vee i \geq \ell_y \vee j \leq 0 \quad S := S, x \approx \epsilon \quad || \\ A := A, 0 \leq i < \ell_y, j > 0, \ell_k \approx i, \ell_x \approx \min(j, \ell_y - i) \\ S := S, y \approx k ++ x ++ k' \end{array}} \\
\\
\text{R-Nth} \frac{x \approx \text{nth}(y, i) \in S}{\begin{array}{l} A := A, i < 0 \vee i \geq \ell_y \quad || \\ A := A, 0 \leq i < \ell_y, \ell_k \approx i \quad S := S, y \approx k ++ \text{unit}(x) ++ k' \end{array}} \\
\\
\text{R-Update} \frac{x \approx \text{update}(y, i, z) \in S}{\begin{array}{l} A := A, i < 0 \vee i \geq \ell_y \quad S := S, x \approx y \quad || \\ A := A, 0 \leq i < \ell_y, \ell_k \approx i, \ell_{k'} \approx 1 \\ S := S, y \approx k ++ k' ++ k'', x \approx k ++ \text{unit}(z) ++ k'' \end{array}}
\end{array}$$

**Fig. 5.** Reduction rules for extract, nth, and update. The rules use  $k$ ,  $k'$ , and  $k''$  to denote fresh sequence variables. We write  $s \approx \min(t, u)$  as an abbreviation for  $s \approx t \vee s \approx u, s \leq t, s \leq u$ .

Finally, Deq-Ext handles disequalities between sequences  $x$  and  $y$  by either asserting that their lengths are different or by choosing an index  $i$  at which they differ.

Figure 5 includes a set of reduction rules for handling operators that are not directly handled by the core rules. These reduction rules capture the semantics of these operators by reduction to concatenation. R-Extract splits into two cases: Either the extraction uses an out-of-bounds index or a non-positive length, in which case the result is the empty sequence, or the original sequence can be described as a concatenation that includes the extracted sub-sequence. R-Nth creates an equation between  $y$  and a concatenation term with  $\text{unit}(x)$  as one of its components, as long as  $i$  is not out of bounds. R-Update considers two cases. If  $i$  is out of bounds, then the update term is equal to  $y$ . Otherwise,  $y$  is equal to a concatenation, with the middle component ( $k'$ ) representing the part of  $y$  that is updated. In the update term,  $k'$  is replaced by  $\text{unit}(z)$ .

*Example 4.* Consider a configuration  $(S, A)$ , where  $S$  contains the formulas  $x \approx y ++ z$ ,  $z \approx v ++ x ++ w$ , and  $v \approx \text{unit}(u)$ , and  $A$  is empty. Hence,  $S \models |x| \approx |y ++ z|$ . By L-Intro, we have  $S \models |y ++ z| \approx |y| + |z|$ . Together with Assumption 1, we have  $S \models \ell_x \approx \ell_y + \ell_z$ , and then with S-Prop, we have  $\ell_x \approx \ell_y + \ell_z \in A$ . Similarly, we can derive  $\ell_z \approx \ell_v + \ell_x + \ell_w, \ell_v \approx 1 \in S$ , and so  $(*)A \models_{\text{LIA}} \ell_z \approx 1 + \ell_y + \ell_z + \ell_w$ . Notice that for any variable  $k$  of sort Seq, we can apply L-Valid, L-Intro, and S-Prop to add to  $A$  either  $\ell_k > 0$  or  $\ell_k = 0$ . Applying this to  $y, z, w$ , we have that  $A \models_{\text{LIA}} \perp$  in each branch thanks to  $(*)$ , and so A-Conf applies and we get unsat.

## 4.2 Extended Calculus

**Definition 9.** The calculus EXT is comprised of the derivation rules in Figs. 4 and 6, with the addition of rule R-Extract from Fig. 5.

Our extended calculus combines array reasoning, based on [10] and expressed by the rules in Fig. 6, with the core rules of Fig. 4 and the R-Extract rule. Unlike in BASE, those rules do not reduce nth and update. Instead, they reason about those operators directly and handle their combination with concatenation. Nth-Concat identifies the  $i$ th

element of sequence  $y$  with the corresponding element selected from its normal form (see Lemma 4). Update-Concat operates similarly, applying update to all the components. Update-Concat-Inv operates similarly on the updated sequence rather than on the original sequence. Nth-Unit captures the semantics of  $\text{nth}$  when applied to a unit term. Update-Unit is similar and distinguishes an update on an out-of-bounds index (different from 0) from an update within the bound. Nth-Intro is meant to ensure that Nth-Update (explained below) and Nth-Unit (explained above) are applicable whenever an update term exists in the constraints. Nth-Update captures the read-over-write axioms of arrays, adapted to consider their lengths (see, e.g., [10]). It distinguishes three cases: In the first, the update index is out of bounds. In the second, it is not out of bounds, and the corresponding  $\text{nth}$  term accesses the same index that was updated. In the third case, the index used in the  $\text{nth}$  term is different from the updated index. Update-Bound considers two cases: either the update changes the sequence, or the sequence remains the same. Finally, Nth-Split introduces a case split on the equality between two sequence variables  $x$  and  $x'$  whenever they appear as arguments to  $\text{nth}$  with equivalent second arguments. This is needed to ensure that we detect all cases where the arguments of two  $\text{nth}$  terms must be equal.

### 4.3 Correctness

In this section we prove the following theorem:

**Theorem 1.** *Let  $X \in \{\text{BASE}, \text{EXT}\}$  and  $(S_0, A_0)$  be a configuration, and assume without loss of generality that  $A_0$  contains only arithmetic constraints that are not sequence constraints. Let  $T$  be a derivation tree obtained by applying the rules of  $X$  with  $(S_0, A_0)$  as the initial configuration.*

1. *If  $T$  is closed, then  $S_0 \cup A_0$  is  $T_{\text{Seq}}$ -unsatisfiable.*
2. *If  $T$  contains a saturated configuration  $(S, A)$  w.r.t.  $X$ , then  $(S, A)$  is  $T_{\text{Seq}}$ -satisfiable, and so is  $(S_0, A_0)$ .*

The theorem states that the calculi are correct in the following sense: if a closed derivation tree is obtained for the constraints  $S_0 \cup A_0$  then those constraints are unsatisfiable in  $T_{\text{Seq}}$ ; if a tree with a saturated leaf is obtained, then they are satisfiable. It is possible, however, that neither kind of tree can be derived by the calculi, making them neither refutation-complete nor terminating. This is not surprising since, as mentioned in the introduction, the decidability of even weaker theories is still unknown.

Proving the first claim in Theorem 1 reduces to a local soundness argument for each of the rules. For the second claim, we sketch below how to construct a satisfying model  $\mathcal{M}$  from a saturated configuration for the case of EXT. The case for BASE is similar and simpler.

*Model Construction Steps.* The full model construction and its correctness are described in a longer version of this paper [23] together with a proof of the theorem above. Here is a summary of the steps needed for the model construction.

1. Sorts:  $\mathcal{M}(\text{Elem})$  is interpreted as some arbitrary countably infinite set.  $\mathcal{M}(\text{Seq})$  and  $\mathcal{M}(\text{Int})$  are then determined by the theory.

$$\begin{array}{c}
\text{Nth-Concat} \frac{x \approx \text{nth}(y, i) \in S \quad S \models_{++}^* y \approx w_1 ++ \dots ++ w_n}{\begin{array}{l} A := A, i < 0 \vee i \geq \ell_y \quad \parallel \\ A := A, 0 \leq i < \ell_{w_1} \quad S := S, x \approx \text{nth}(w_1, i) \quad \parallel \quad \dots \quad \parallel \\ A := A, \sum_{j=1}^{n-1} \ell_{w_j} \leq i < \sum_{j=1}^n \ell_{w_j} \quad S := S, x \approx \text{nth}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j}) \end{array}} \\
\\
\text{Update-Concat} \frac{x \approx \text{update}(y, i, v) \in S \quad S \models_{++}^* y \approx w_1 ++ \dots ++ w_n}{\begin{array}{l} S := S, x \approx z_1 ++ \dots ++ z_n, \\ z_1 \approx \text{update}(w_1, i, v), \dots, z_n \approx \text{update}(w_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v) \end{array}} \\
\\
\text{Update-Concat-Inv} \frac{x \approx \text{update}(y, i, v) \in S \quad S \models_{++}^* x \approx w_1 ++ \dots ++ w_n}{\begin{array}{l} S := S, y \approx z_1 ++ \dots ++ z_n, \\ w_1 \approx \text{update}(z_1, i, v), \dots, w_n \approx \text{update}(z_n, i - \sum_{j=1}^{n-1} \ell_{w_j}, v) \end{array}} \\
\\
\text{Nth-Unit} \frac{x \approx \text{nth}(y, i) \in S \quad S \models y \approx \text{unit}(u)}{A := A, i < 0 \vee i > 0 \quad \parallel \quad A := A, i \approx 0 \quad S := S, x \approx u} \\
\\
\text{Update-Unit} \frac{x \approx \text{update}(y, i, v) \in S \quad S \models y \approx \text{unit}(u)}{\begin{array}{l} A := A, i < 0 \vee i > 0 \quad S := S, x \approx \text{unit}(u) \quad \parallel \\ A := A, i \approx 0 \quad S := S, x \approx \text{unit}(v) \end{array}} \\
\\
\text{Nth-Intro} \frac{s' \approx \text{update}(s, i, t) \in S}{S := S, e \approx \text{nth}(s, i), e' \approx \text{nth}(s', i)} \\
\\
\text{Nth-Update} \frac{\text{nth}(x, j) \in \mathcal{T}(S) \quad y \approx \text{update}(z, i, v) \in S \quad S \models x \approx y \text{ or } S \models x \approx z}{\begin{array}{l} A := A, j < 0 \vee j \geq \ell_x \quad \parallel \\ A := A, i \approx j, 0 \leq j < \ell_x \quad S := S, \text{nth}(y, j) \approx v \quad \parallel \\ A := A, i \not\approx j, 0 \leq j < \ell_x \quad S := S, \text{nth}(y, j) \approx \text{nth}(z, j) \end{array}} \\
\\
\text{Update-Bound} \frac{x \approx \text{update}(y, i, v) \in S}{A := A, 0 \leq i < \ell_y \quad S := S, \text{nth}(y, i) \not\approx v \quad \parallel \quad S := S, x \approx y} \\
\\
\text{Nth-Split} \frac{\text{nth}(x, i), \text{nth}(x', i') \in \mathcal{T}(S) \quad i \approx i' \in A}{S := S, x \approx x' \quad \parallel \quad S := S, x \not\approx x'}
\end{array}$$

**Fig. 6.** Extended derivation rules. The rules use  $z_1, \dots, z_n$  to denote fresh sequence variables and  $e, e'$  to denote fresh element variables.

2.  $\Sigma_{\text{Seq}}$ -symbols:  $T_{\text{Seq}}$  enforces the interpretation of almost all  $\Sigma_{\text{Seq}}$ -symbols, except for  $\text{nth}$  when the second input is out of bounds. We cover this case below.
3. Integer variables: based on the saturation of  $\mathbf{A}\text{-Conf}$ , we know there is some  $T_{\text{LIA}}$ -model satisfying  $A$ . We set  $\mathcal{M}$  to interpret integer variables according to this model.
4. Element variables: these are partitioned into their  $\equiv_S$  equivalence classes. Each class is assigned a distinct element from  $\mathcal{M}(\text{Elem})$ , which is possible since it is infinite.
5. Atomic sequence variables: these are assigned interpretations in several sub-steps:
  - (a) length: we first use the assignments to variables  $\ell_x$  to set the length of  $\mathcal{M}(x)$ , without assigning its actual value.
  - (b) unit variables: for variables  $x$  with  $x \equiv_S \text{unit}(z)$ , we set  $\mathcal{M}(x)$  to be  $[\mathcal{M}(z)]$ .



- (c) non-unit variables: All other sequence variables are assigned values according to a *weak equivalence graph* we construct in a manner similar to [10]. This construction takes into account constraints that involve update and nth.
- 6. Non-atomic sequence variables: these are first transformed to their unique normal form (see Lemma 4), consisting of concatenations of atomic variables. Then, the values assigned to these variables are concatenated.
- 7. nth-terms: for out-of-bounds indices in nth-terms, we rely on  $\equiv_5$  to make sure that the assignment is consistent.

We conclude this section with an example of the construction of  $\mathcal{M}$ .

*Example 5.* Consider a signature in which Elem is Int, and a saturated configuration  $(S^*, A^*)$  w.r.t. EXT that includes the following formulas:  $y \approx y_1 ++ y_2$ ,  $x \approx x_1 ++ x_2$ ,  $y_2 \approx x_2$ ,  $y_1 \approx \text{update}(x_1, i, a)$ ,  $|y_1| = |x_1|$ ,  $|y_2| = |x_2|$ ,  $\text{nth}(y, i) \approx a$ ,  $\text{nth}(y_1, i) \approx a$ . Following the above construction, a satisfying interpretation  $\mathcal{M}$  can be built as follows:

- Step 1** Set both  $\mathcal{M}(\text{Int})$  and  $\mathcal{M}(\text{Elem})$  to be the set of integer numbers.  $\mathcal{M}(\text{Seq})$  is fixed by the theory.
- Step 3, Step 4** First, find an arithmetic model,  $\mathcal{M}(\ell_x) = \mathcal{M}(\ell_y) = 4$ ,  $\mathcal{M}(\ell_{y_1}) = \mathcal{M}(\ell_{x_1}) = 2$ ,  $\mathcal{M}(\ell_{y_2}) = \mathcal{M}(\ell_{x_2}) = 2$ ,  $\mathcal{M}(i) = 0$ . Further, set  $\mathcal{M}(a) = 0$ .
- Step 5a** Start assigning values to sequences. First, set the lengths of  $\mathcal{M}(x)$  and  $\mathcal{M}(y)$  to be 4, and the lengths of  $\mathcal{M}(x_1)$ ,  $\mathcal{M}(x_2)$ ,  $\mathcal{M}(y_1)$ ,  $\mathcal{M}(y_2)$  to be 2.
- Step 5b** is skipped as there are no unit terms.
- Step 5c** Set the 0th element of  $\mathcal{M}(y_1)$  to 0 to satisfy  $\text{nth}(y_1, i) = a$  ( $y_1$  is atomic,  $y$  is not). Assign fresh values to the remaining indices of atomic variables. The result can be, e.g.,  $\mathcal{M}(y_1) = [0, 2]$ ,  $\mathcal{M}(x_1) = [1, 2]$ ,  $\mathcal{M}(y_2) = \mathcal{M}(x_2) = [3, 4]$ .
- Step 6** Assign non-atomic sequence variables based on equivalent concatenations:  $\mathcal{M}(y) = [0, 2, 3, 4]$ ,  $\mathcal{M}(x) = [1, 2, 3, 4]$ .
- Step 7** No integer variable in the formula was assigned an out-of-bound value, and so the interpretation of nth on out-of-bounds cases is set arbitrarily.

## 5 Implementation

We implemented our procedure for sequences as an extension of a previous theory solver for strings [17, 22]. This solver is integrated in cvc5, and has been generalized to reason about both strings and sequences. In this section, we describe how the rules of the calculus are implemented and the overall strategy for when they are applied.

Like most SMT solvers, cvc5 is based on the CDCL( $T$ ) architecture [19] which combines several subsolvers, each specialized on a specific theory, with a solver for propositional satisfiability (SAT). Following that architecture, cvc5 maintains an evolving set of formulas  $F$ . When  $F$  starts with quantifier-free formulas over the theory  $T_{\text{Seq}}$ , the case targeted by this work, the SAT solver searches for a satisfying assignment for  $F$ , represented as the set  $M$  of literals it satisfies. If none exists, the problem is unsatisfiable at the propositional level and hence  $T_{\text{Seq}}$ -unsatisfiable. Otherwise,  $M$  is partitioned into the arithmetic constraints  $A$  and the sequence constraints  $S$  and checked for  $T_{\text{Seq}}$ -satisfiability using the rules of the EXT calculus. Many of those rules, including all

those with multiple conclusions, are implemented by adding new formulas to  $F$  (following the splitting-on-demand approach [4]). This causes the SAT solver to try to extend its assignment to those formulas, which results in the addition of new literals to  $M$  (and thereby also to  $A$  and  $S$ ).

In this setting, the rules of the two calculi are implemented as follows. The effect of rule A-Conf is achieved by invoking *cvc5*'s theory solver for linear integer arithmetic. Rule S-Conf is implemented by the congruence closure submodule of the theory solver for sequences. Rules A-Prop and S-Prop are implemented by the standard mechanism for theory combination. Note that each of these four rules may be applied *eagerly*, that is, before constructing a complete satisfying assignment  $M$  for  $F$ .

The remaining rules are implemented in the theory solver for sequences. Each time  $M$  is checked for satisfiability, *cvc5* follows a strategy to determine which rule to apply next. If none of the rules apply and the configuration is different from *unsat*, then it is saturated, and the solver returns *sat*. The strategy for EXT prioritizes rules as follows. Only the first applicable rule is applied (and then control goes back to the SAT solver).

1. (Add length constraints) For each sequence term in  $S$ , apply L-Intro or L-Valid, if not already done. We apply L-Intro for non-variables, and L-Valid for variables.
2. (Mark congruent terms) For each set of update (resp. nth) terms that are congruent to one another in the current configuration, mark all but one term and ignore the marked terms in the subsequent steps.
3. (Reduce extract) For  $\text{extract}(y, i, j)$  in  $S$ , apply R-Extract if not already done.
4. (Construct normal forms) Apply U-Eq or C-Split. We choose how to apply the latter rule based on constructing normal forms for equivalence classes in a bottom-up fashion, where the equivalence classes of  $x$  and  $y$  are considered before the equivalence class of  $x ++ y$ . We do this until we find an equivalence class such that  $S \models_{++}^* z \approx u_1$  and  $S \models_{++}^* z \approx u_2$  for distinct  $u_1, u_2$ .
5. (Normal forms) Apply C-Eq if two equivalence classes have the same normal form.
6. (Extensionality) For each disequality in  $S$ , apply Deq-Ext, if not already done.
7. (Distribute update and nth) For each term  $\text{update}(x, i, t)$  (resp.  $\text{nth}(x, j)$ ) such that the normal form of  $x$  is a concatenation term, apply Update-Concat and Update-Concat-Inv (resp. Nth-Concat) if not already done. Alternatively, if the normal form of the equivalence class of  $x$  is a unit term, apply Update-Unit (resp. Nth-Unit).
8. (Array reasoning on atomic sequences) Apply Nth-Intro and Update-Bound to update terms. For each update term, find the matching nth terms and apply Nth-Update. Apply Nth-Split to pairs of nth terms with equivalent indices.
9. (Theory combination) Apply S-A for all arithmetic terms occurring in both  $S$  and  $A$ .

Whenever a rule is applied, the strategy will restart from the beginning in the next iteration. The strategy is designed to apply with higher priority steps that are easy to compute and are likely to lead to conflicts. Some steps are ordered based on dependencies from other steps. For instance, Steps 5 and 7 use normal forms, which are computed in Step 4. The strategy for the BASE calculus is the same, except that Steps 7 and 8 are replaced by one that applies R-Update and R-Nth to all update and nth terms in  $S$ .

We point out that the C-Split rule may cause non-termination of the proof strategy described above in the presence of *cyclic* sequence constraints, for instance, constraints where sequence variables appear on both sides of an equality. The solver uses methods

for detecting some of these cycles, to restrict when C-Split is applied. In particular, when  $S \models_{++}^* x \approx (\bar{u} ++ s ++ \bar{w}) \downarrow$ ,  $S \models_{++}^* x \approx (\bar{u} ++ t ++ \bar{v}) \downarrow$ , and  $s$  occurs in  $\bar{v}$ , then C-Split is not applied. Instead, other heuristics are used, and in some cases the solver terminates with a response of “unknown” (see e.g., [17] for details). In addition to the version shown here, we also use another variation of the C-Split rule where the normal forms are matched in reverse (starting from the last terms in the concatenations). The implementation also uses fast entailment tests for length inequalities. These tests may allow us to conclude which branch of C-Split, if any, is feasible, without having to branch on cases explicitly.

Although not shown here, the calculus can also accommodate certain *extended* sequence constraints, that is, constraints using a signature with additional functions. For example, our implementation supports sequence containment, replacement, and reverse. It also supports an extended variant of the update operator, in which the third argument is a sequence that overrides the sequence being updated starting from the index given in the second argument. Constraints involving these functions are handled by reduction rules, similar to those shown in Fig. 5. The implementation is further optimized by using context-dependent simplifications, which may eagerly infer when certain sequence terms can be simplified to constants based on the current set of assertions [22].

## 6 Evaluation

We evaluate the performance of our approach, as implemented in *cvc5*. The evaluation investigates: (i) whether the use of sequences is a viable option for reasoning about vectors in programs, (ii) how our approach compares with other sequence solvers, and (iii) what is the performance impact of our array-style extended rules. As a baseline, we use Version 4.8.14 of the Z3 SMT solver, which supports a theory of sequences without updates. For *cvc5*, we evaluate implementations of both the basic calculus (denoted **cvc5**) and the extended array-based calculus (denoted **cvc5-a**). The benchmarks, solver configurations, and logs from our runs are available for download.<sup>4</sup> We ran all experiments on a cluster equipped with Intel Xeon E5-2620 v4 CPUs. We allocated one physical CPU core and 8 GB of RAM for each solver-benchmark pair and used a time limit of 300 s. We use the following two sets of benchmarks:

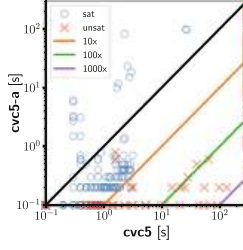
**Array Benchmarks (ARRAYS).** The first set of benchmarks is derived from the QF\_AX benchmarks in SMT-LIB [3]. To generate these benchmarks, we (i) replace declarations of arrays with declarations of sequences of uninterpreted sorts, (ii) change the sort of index terms to integers, and (iii) replace store with update and select with nth. The resulting benchmarks are quantifier-free and do not contain concatenations. Note that the original and the derived benchmarks are not equisatisfiable, because sequences take into account out-of-bounds cases that do not occur in arrays. For the Z3 runs, we add to the benchmarks a definition of update in terms of extraction and concatenation.

**Smart Contract Verification (DIEM).** The second set of benchmarks consists of verification conditions generated by running the Move Prover [24] on smart contracts written for the Diem framework. By default, the encoding does not use the sequence update

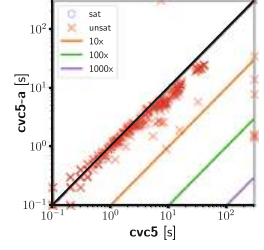
<sup>4</sup> <http://dx.doi.org/10.5281/zenodo.6146565>.

Set		w/ update		Z3
		cvc5	cvc5-a	
ARRAYS (551)	Slvd	242	<b>390</b>	170
	Time	162	303	4329
DIEM (558)	Slvd	542	<b>547</b>	443
	Time	518	440	639

(a)



(b) ARRAYS



(c) DIEM

**Fig. 7.** Figure a lists the number of solved benchmarks and total time on commonly solved benchmarks. The scatter plots compare the base solver (**cvc5**) and the extended solver (**cvc5-a**) on ARRAY (Fig. b) and DIEM (Fig. c) benchmarks.

operation, and so Z3 can be used directly. However, we also modified the Move Prover encoding to generate benchmarks that do use the update operator, and ran **cvc5** on them. In addition to using the sequence theory, the benchmarks make heavy use of quantifiers and the SMT-LIB theory of datatypes.

Figure 7a summarizes the results in terms of number of solved benchmarks and total time in seconds on commonly solved benchmarks. The configuration that solves the largest number of benchmarks is the implementation of the extended calculus (**cvc5-a**). This approach also successfully solves most of the DIEM benchmarks, which suggests that sequences are a promising option for encoding vectors in programs. The results further show that the sequences solver of **cvc5** significantly outperforms Z3 on both the number of solved benchmarks and the solving time on commonly solved benchmarks.

Figures 7b and 7c show scatter plots comparing **cvc5** and **cvc5-a** on the two benchmark sets. We can see a clear trend towards better performance when using the extended solver. In particular, the table shows that in addition to solving the most benchmarks, **cvc5-a** is also fastest on the commonly solved instances from the DIEM benchmark set.

For the ARRAYS set, we can see that some benchmarks are slower with the extended solver. This is also reflected in the table, where **cvc5-a** is slower on the commonly solved instances. This is not too surprising, as the extra machinery of the extended solver can sometimes slow down easy problems. As problems get harder, however, the benefit of the extended solver becomes clear. For example, if we drop Z3 and consider just the commonly solved instances between **cvc5** and **cvc5-a** (of which there are 242), **cvc5-a** is about  $2.47\times$  faster (426 vs 1053 s). Of course, further improving the performance of **cvc5-a** is something we plan to explore in future work.

## 7 Conclusion

We introduced calculi for checking satisfiability in the theory of sequences, which can be used to model the vector data type. We described our implementation in **cvc5** and provided an evaluation, showing that the proposed theory is rich enough to naturally

express verification conditions without introducing quantifiers, and that our implementation is efficient. We believe that verification tools can benefit by changing their encoding of verification conditions that involve vectors to use the proposed theory and implementation.

We plan to propose the incorporation of this theory in the SMT-LIB standard and contribute our benchmarks to SMT-LIB. As future research, we plan to integrate other approaches for array solving into our basic solver. We also plan to study the politeness [16, 20] and decidability of various fragments of the theory of sequences.

## References

1. Alberti, F., Ghilardi, S., Pagani, E.: Cardinality constraints for arrays (decidability results and applications). *Formal Methods Syst. Des.* **51**(3), 545–574 (2017). <https://doi.org/10.1007/s10703-017-0279-6>
2. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) *TACAS 2022*. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-99524-9\\_24](https://doi.org/10.1007/978-3-030-99524-9_24)
3. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa (2017). [www.SMT-LIB.org](http://www.SMT-LIB.org)
4. Barrett, C., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Splitting on demand in SAT modulo theories. In: Hermann, M., Voronkov, A. (eds.) *LPAR 2006*. LNCS (LNAI), vol. 4246, pp. 512–526. Springer, Heidelberg (2006). [https://doi.org/10.1007/11916277\\_35](https://doi.org/10.1007/11916277_35)
5. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) *Handbook of Model Checking*, pp. 305–343. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-10575-8\\_11](https://doi.org/10.1007/978-3-319-10575-8_11)
6. Berzish, M., Ganesh, V., Zheng, Y.: Z3str3: a string solver with theory-aware heuristics. In: Stewart, D., Weissenbacher, G. (eds.) *2017 Formal Methods in Computer Aided Design, FMCAD 2017*, Vienna, Austria, 2–6 October 2017, pp. 55–59. IEEE (2017)
7. Bjørner, N., de Moura, L., Nachmanson, L., Wintersteiger, C.: Programming Z3 (2018). <https://theory.stanford.edu/~nikolaj/programmingz3.html#sec-sequences-and-strings>
8. Bjørner, N., Ganesh, V., Michel, R., Veanes, M.: An SMT-LIB format for sequences and regular expressions. *SMT* **12**, 76–86 (2012)
9. Bjørner, N., Tillmann, N., Voronkov, A.: Path feasibility analysis for string-manipulating programs. In: Kowalewski, S., Philippou, A. (eds.) *TACAS 2009*. LNCS, vol. 5505, pp. 307–321. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-00768-2\\_27](https://doi.org/10.1007/978-3-642-00768-2_27)
10. Christ, J., Hoenicke, J.: Weakly equivalent arrays. In: Lutz, C., Ranise, S. (eds.) *FroCoS 2015*. LNCS (LNAI), vol. 9322, pp. 119–134. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-24246-0\\_8](https://doi.org/10.1007/978-3-319-24246-0_8)
11. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) *TACAS 2008*. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_24](https://doi.org/10.1007/978-3-540-78800-3_24)
12. Elad, N., Rain, S., Immerman, N., Kovács, L., Sagiv, M.: Summing up smart transitions. In: Silva, A., Leino, K.R.M. (eds.) *CAV 2021*. LNCS, vol. 12759, pp. 317–340. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-81685-8\\_15](https://doi.org/10.1007/978-3-030-81685-8_15)
13. Enderton, H.B.: *A Mathematical Introduction to Logic*, 2nd edn. Academic Press (2001)
14. Falke, S., Merz, F., Sinz, C.: Extending the theory of arrays: `memset`, `memcpy`, and beyond. In: Cohen, E., Rybalchenko, A. (eds.) *VSTTE 2013*. LNCS, vol. 8164, pp. 108–128. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-54108-7\\_6](https://doi.org/10.1007/978-3-642-54108-7_6)

15. Ganesh, V., Minnes, M., Solar-Lezama, A., Rinard, M.: Word equations with length constraints: what's decidable? In: Biere, A., Nahir, A., Vos, T. (eds.) HVC 2012. LNCS, vol. 7857, pp. 209–226. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39611-3\\_21](https://doi.org/10.1007/978-3-642-39611-3_21)
16. Jovanović, D., Barrett, C.: Polite theories revisited. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 402–416. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-16242-8\\_29](https://doi.org/10.1007/978-3-642-16242-8_29)
17. Liang, T., Reynolds, A., Tinelli, C., Barrett, C., Deters, M.: A DPLL( $T$ ) theory solver for a theory of strings and regular expressions. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 646–662. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08867-9\\_43](https://doi.org/10.1007/978-3-319-08867-9_43)
18. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979)
19. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL( $T$ ). J. ACM **53**(6), 937–977 (2006)
20. Ranise, S., Ringeissen, C., Zarba, C.G.: Combining data structures with nonstably infinite theories using many-sorted logic. In: Gramlich, B. (ed.) FroCoS 2005. LNCS (LNAI), vol. 3717, pp. 48–64. Springer, Heidelberg (2005). [https://doi.org/10.1007/11559306\\_3](https://doi.org/10.1007/11559306_3)
21. Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.: Reductions for strings and regular expressions revisited. In: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, 21–24 September 2020, pp. 225–235. IEEE (2020)
22. Reynolds, A., Woo, M., Barrett, C., Brumley, D., Liang, T., Tinelli, C.: Scaling up DPLL( $T$ ) string solvers using context-dependent simplification. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 453–474. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_24](https://doi.org/10.1007/978-3-319-63390-9_24)
23. Sheng, Y., et al.: Reasoning about vectors using an SMT theory of sequences. CoRR 10.48550/ARXIV.2205.08095 (2022)
24. Zhong, J.E., et al.: The move prover. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 137–150. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-53288-8\\_7](https://doi.org/10.1007/978-3-030-53288-8_7)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



# **Calculi and Orderings**



# An Efficient Subsumption Test Pipeline for BS(LRA) Clauses

Martin Bromberger<sup>1</sup> , Lorenz Leutgeb<sup>1,2</sup> , and Christoph Weidenbach<sup>1</sup>

<sup>1</sup> Max Planck Institute for Informatics, Saarland Informatics Campus,  
Saarbrücken, Germany

{mbromberger,lorenz,weidenbach}@mpi-inf.mpg.de

<sup>2</sup> Graduate School of Computer Science, Saarland Informatics Campus,  
Saarbrücken, Germany

**Abstract.** The importance of subsumption testing for redundancy elimination in first-order logic automatic reasoning is well-known. Although the problem is already NP-complete for first-order clauses, the meanwhile developed test pipelines efficiently decide subsumption in almost all practical cases. We consider subsumption between first-order clauses of the Bernays-Schönfinkel fragment over linear real arithmetic constraints: BS(LRA). The bottleneck in this setup is deciding implication between the LRA constraints of two clauses. Our new *sample point heuristic* pre-empts expensive implication decisions in about 94% of all cases in benchmarks. Combined with filtering techniques for the first-order BS part of clauses, it results again in an efficient subsumption test pipeline for BS(LRA) clauses.

**Keywords:** Bernays-Schönfinkel fragment · Linear arithmetic · Redundancy elimination · Subsumption

## 1 Introduction

The elimination of redundant clauses is crucial for the efficient automatic reasoning in first-order logic. In a resolution [5, 50] or superposition setting [4, 44], a newly inferred clause might be subsumed by a clause that is already known (*forward subsumption*) or it might subsume a known clause (*backward subsumption*). Although the SCL calculi family [1, 11, 21] does not require forward subsumption tests, a property also inherent to the propositional CDCL (Conflict Driven Clause Learning) approach [8, 34, 41, 55, 63], backward subsumption and hence subsumption remains an important test in order to remove redundant clauses.

In this work we present advances in deciding subsumption for constrained clauses, specifically employing the Bernays-Schönfinkel fragment as foreground logic, and linear real arithmetic as background theory, BS(LRA). BS(LRA) is of particular interest because it can be used to model *supervisors*, i.e., components in technical systems that control system functionality. An example for a supervisor is the electronic control unit of a combustion engine. The logics we use



to model supervisors and their properties are called *SupERLogs*—(Sup)ervisor (E)ffective(R)easoning (Log)ics. SupERLogs are instances of function-free first-order logic extended with arithmetic [18], which means BS(LRA) is an example of a SupERLog.

Subsumption is an important redundancy criterion in the context of hierarchic clausal reasoning [6, 11, 20, 35, 37]. At the heart of this paper is a new technique to speed up the treatment of linear arithmetic constraints as part of deciding subsumption. For every clause, we store a solution of its associated constraints, which is used to quickly falsify implication decisions, acting as a filter, called the *sample point heuristic*. In our experiments with various benchmarks, the technique is very effective: It successfully preempts expensive implication decisions in about 94% of cases. We elaborate on these findings in Sect. 4.

For example, consider three BS clauses, none of which subsumes another:

$$C_1 := P(a, x) \quad C_2 := \neg P(y, z) \vee Q(y, z, b) \quad C_3 := \neg R(b) \vee Q(a, x, b)$$

Let  $C_4$  be the resolvent of  $C_1$  and  $C_2$  upon the atom  $P(a, x)$ , i.e.,  $C_4 := Q(a, z, b)$ . Now  $C_4$  backward-subsumes  $C_3$  with matcher  $\sigma := \{z \mapsto x\}$ , i.e.  $C_4\sigma \subset C_3$ , thus  $C_3$  is redundant and can be eliminated. Now, consider an extension of the above clauses with some simple LRA constraints following the same reasoning:

$$\begin{aligned} C'_1 &:= x \geq 1 \parallel P(a, x) \\ C'_2 &:= z \geq 0 \parallel \neg P(y, z) \vee Q(y, z, b) \\ C'_3 &:= x \geq 0 \parallel \neg R(b) \vee Q(a, x, b) \end{aligned}$$

where  $\parallel$  is interpreted as an implication, i.e., clause  $C'_1$  stands for  $\neg x \geq 1 \vee P(a, x)$  or simply  $x < 1 \vee P(a, x)$ . The respective resolvent on the constrained clauses is  $C'_4 := z \geq 0, z \geq 1 \parallel Q(a, z, b)$  or after constraint simplification  $C'_4 := z \geq 1 \parallel Q(a, z, b)$  because  $z \geq 1$  implies  $z \geq 0$ . For the constrained clauses,  $C'_4$  does no longer subsume  $C'_3$  with matcher  $\sigma := \{z \mapsto x\}$ , because  $z \geq 0$  does not LRA-imply  $z \geq 1$ . Now, if we store the sample point  $x = 0$  as a solution for the constraint of clause  $C'_3$ , this sample point already reveals that  $z \geq 0$  does not LRA-imply  $z \geq 1$ . This constitutes the basic idea behind our sample point heuristic. In general, constraints are not just simple bounds as in the above example, and sample points are solutions to the system of linear inequalities of the LRA constraint of a clause.

Please note that our test on LRA constraints is based on LRA theory implication and not on a syntactic notion such as subsumption on the first-order part of the clause. In this sense it is “stronger” than its first-order counterpart. This fact is stressed by the following example, taken from [26, Ex. 2], which shows that first-order implication does not imply subsumption. Let

$$\begin{aligned} C_1 &:= \neg P(x, y) \vee \neg P(y, z) \vee P(x, z) \\ C_2 &:= \neg P(a, b) \vee \neg P(b, c) \vee \neg P(c, d) \vee P(a, d) \end{aligned}$$

Then we have  $C_1 \rightarrow C_2$ , but again, for all  $\sigma$  we have  $C_1\sigma \not\subseteq C_2$ : Constructing  $\sigma$  from left to right we obtain  $\sigma := \{x \mapsto a, y \mapsto b, z \mapsto c\}$ , but  $P(a, c) \notin C_2$ .

Constructing  $\sigma$  from right to left we obtain  $\sigma := \{z \mapsto d, x \mapsto a, y \mapsto c\}$ , but  $\neg P(a, c) \notin C_2$ .

*Related Work.* Treatment of questions regarding the complexity of deciding subsumption of first-order clauses [27] dates back more than thirty years. Notions of subsumption, varying in generality, are studied in different sub-fields of theorem proving, whereas we restrict our attention to first-order theorem proving. Modern implementations typically decide multiple thousand instances of this problem per second: In [62, Sect. 2], Voronkov states that initial versions of Vampire “seemed to [...] deadlock” without efficient implementations to decide (forward) subsumption.

In order to reduce the number of clauses out of a set of clauses to be considered for pairwise subsumption checking, the best known practice in first-order theorem proving is to use (imperfect) indexing data structures as a means for pre-filtering and research concerning appropriate techniques is plentiful, see [24, 25, 27–30, 33, 39, 40, 43, 45–49, 52–54, 56, 59, 61] for an evaluation of these techniques. Here we concentrate on the efficiency of a subsumption check between two clauses and therefore do not take indexing techniques into account. Furthermore, the implication test between two linear arithmetic constraints is of a semantic nature and is not related to any syntactic features of the involved constraints and can therefore hardly be filtered by a syntactic indexing approach.

In addition to pre-filtering via indexing, almost all above mentioned implementations of first-order subsumption tests rely on additional filters on the clause level. The idea is to generate an abstraction of clauses together with an ordering relation such that the ordering relation is necessary to hold between two clauses in order for one clause to subsume the other. Furthermore, the abstraction as well as the ordering relation should be efficiently computable. For example, a necessary condition for a first-order clause  $C_1$  to subsume a first-order clause  $C_2$  is  $|\text{vars}(C_1)| \geq |\text{vars}(C_2)|$ , i.e., the number of different variables in  $C_1$  must be larger or equal than the number of variables in  $C_2$ . Further and additional abstractions included by various implementations rely on the size of clauses, number of ground literals, depth of literals and terms, occurring predicate and function symbols. For the BS(LRA) clauses considered here, the structure of the first-order BS part, which consists of predicates and flat terms (variables and constants) only, is not particularly rich.

The exploration of sample points has already been studied in the context of first-order clauses with arithmetic constraints. In [17, 36] it was used to improve the performance of iSAT [23] on testing non-linear arithmetic constraints. In general, iSAT tests satisfiability by interval propagation for variables. If intervals get “too small” it typically gives up, however sometimes the explicit generation of a sample point for a small interval can still lead to a certificate for satisfiability. This technique was successfully applied in [17], but was not used for deciding subsumption of constrained clauses.

*Motivation.* The main motivation for this work is the realization that computing implication decisions required to treat constraints of the background theory presents the bottleneck of an BS(LRA) subsumption check in practice. Inspired by the success of filtering techniques in first-order logic, we devise an exceptionally effective filter for constraints and adopt well-known first-order filters to the BS fragment. Our sample point heuristic for LRA could easily be generalized to other arithmetic theories as well as full first-order logic.

*Structure.* The paper is structured as follows. After a section defining BS(LRA) and common notions and notation, Sect. 2, we define redundancy notions and our sample point heuristic in Sect. 3. Section 4 justifies the success of the sample point heuristic by numerous experiments in various application domains of BS(LRA). The paper ends with a discussion of the obtained results, Sect. 5. Binaries, utility scripts, benchmarking instances used as input, and the output used for evaluation may be obtained online [13].

## 2 Preliminaries

We briefly recall the basic logical formalisms and notations we build upon [10]. Our starting point is a standard many-sorted first-order language for BS with *constants* (denoted  $a, b, c$ ), without non-constant function symbols, with *variables* (denoted  $w, x, y, z$ ), and *predicates* (denoted  $P, Q, R$ ) of some fixed *arity*. *Terms* (denoted  $t, s$ ) are variables or constants. An *atom* (denoted  $A, B$ ) is an expression  $P(t_1, \dots, t_n)$  for a predicate  $P$  of arity  $n$ . A *positive literal* is an atom  $A$  and a *negative literal* is a negated atom  $\neg A$ . We define  $\text{comp}(A) = \neg A$ ,  $\text{comp}(\neg A) = A$ ,  $|A| = A$  and  $|\neg A| = A$ . Literals are usually denoted  $L, K, H$ . Formulas are defined in the usual way using quantifiers  $\forall, \exists$  and the boolean connectives  $\neg, \vee, \wedge, \rightarrow$ , and  $\equiv$ .

A *clause* (denoted  $C, D$ ) is a universally closed disjunction of literals  $A_1 \vee \dots \vee A_n \vee \neg B_1 \vee \dots \vee \neg B_m$ . Clauses are identified with their respective multisets and all standard multiset operations are extended to clauses. For instance,  $C \subseteq D$  means that all literals in  $C$  also appear in  $D$  respecting their number of occurrences. A clause is *Horn* if it contains at most one positive literal, i.e.  $n \leq 1$ , and a *unit clause* if it has exactly one literal, i.e.  $n + m = 1$ . We write  $C^+$  for the set of positive literals, or *conclusions* of  $C$ , i.e.  $C^+ := \{A_1, \dots, A_n\}$  and respectively  $C^-$  for the set of negative literals, or *premises* of  $C$ , i.e.  $C^- := \{\neg B_1, \dots, \neg B_m\}$ . If  $Y$  is a term, formula, or a set thereof,  $\text{vars}(Y)$  denotes the set of all variables in  $Y$ , and  $Y$  is *ground* if  $\text{vars}(Y) = \emptyset$ .

The *Bernays-Schönfinkel Clause Fragment* (BS) in first-order logic consists of first-order clauses where all involved terms are either variables or constants. The *Horn Bernays-Schönfinkel Clause Fragment* (HBS) consists of all sets of BS Horn clauses.

A *substitution*  $\sigma$  is a function from variables to terms with a finite domain  $\text{dom}(\sigma) = \{x \mid x\sigma \neq x\}$  and codomain  $\text{codom}(\sigma) = \{x\sigma \mid x \in \text{dom}(\sigma)\}$ . We denote substitutions by  $\sigma, \delta, \rho$ . The application of substitutions is often written

postfix, as in  $x\sigma$ , and is homomorphically extended to terms, atoms, literals, clauses, and quantifier-free formulas. A substitution  $\sigma$  is *ground* if  $\text{codom}(\sigma)$  is ground. Let  $Y$  denote some term, literal, clause, or clause set. A substitution  $\sigma$  is a *grounding* for  $Y$  if  $Y\sigma$  is ground, and  $Y\sigma$  is a *ground instance* of  $Y$  in this case. We denote by  $\text{gnd}(Y)$  the set of all ground instances of  $Y$ , and by  $\text{gnd}_B(Y)$  the set of all ground instances over a given set of constants  $B$ . The *most general unifier*  $\text{mgu}(Z_1, Z_2)$  of two terms/atoms/literals  $Z_1$  and  $Z_2$  is defined as usual, and we assume that it does not introduce fresh variables and is idempotent.

We assume a standard many-sorted first-order logic model theory, and write  $\mathcal{A} \models \phi$  if an interpretation  $\mathcal{A}$  satisfies a first-order formula  $\phi$ . A formula  $\psi$  is a logical consequence of  $\phi$ , written  $\phi \models \psi$ , if  $\mathcal{A} \models \psi$  for all  $\mathcal{A}$  such that  $\mathcal{A} \models \phi$ . Sets of clauses are semantically treated as conjunctions of clauses with all variables quantified universally.

## 2.1 Bernays-Schönfinkel with Linear Real Arithmetic

The extension of BS with linear real arithmetic, BS(LRA), is the basis for the formalisms studied in this paper. We consider a standard *many-sorted* first-order logic with one first-order sort  $\mathcal{F}$  and with the sort  $\mathcal{R}$  for the real numbers. Given a clause set  $N$ , the interpretations  $\mathcal{A}$  of our sorts are fixed:  $\mathcal{R}^{\mathcal{A}} = \mathbb{R}$  and  $\mathcal{F}^{\mathcal{A}} = \mathbb{F}$ . This means that  $\mathcal{F}^{\mathcal{A}}$  is a Herbrand interpretation, i.e.,  $\mathbb{F}$  is the set of first-order constants in  $N$ , or a single constant out of the signature if no such constant occurs. Note that this is not a deviation from standard semantics in our context as for the arithmetic part the canonical domain is considered and the first-order sort has the finite model property over the occurring constants (note that equality is not part of BS).

Constant symbols, arithmetic function symbols, variables, and predicates are uniquely declared together with their respective sort. The unique sort of a constant symbol, variable, predicate, or term is denoted by the function  $\text{sort}(Y)$  and we assume all terms, atoms, and formulas to be well-sorted. We assume *pure* input clause sets, which means the only constants of sort  $\mathcal{R}$  are (rational) numbers. This means the only constants that we do allow are rational numbers  $c \in \mathbb{Q}$  and the constants defining our finite first-order sort  $\mathcal{F}$ . Irrational numbers are not allowed by the standard definition of the theory. The current implementation comes with the caveat that only integer constants can be parsed. Satisfiability of pure BS(LRA) clause sets is semi-decidable, e.g., using *hierarchical superposition* [6] or *SCL(T)* [11]. Impure BS(LRA) is no longer compact and satisfiability becomes undecidable, but its restriction to ground clause sets is decidable [22].

All arithmetic predicates and functions are interpreted in the usual way. An interpretation of BS(LRA) coincides with  $\mathcal{A}^{\text{LRA}}$  on arithmetic predicates and functions, and freely interprets free predicates. For pure clause sets this is well-defined [6]. Logical satisfaction and entailment is defined as usual, and uses similar notation as for BS.

*Example 1.* The clause  $y < 5 \vee x' \neq x + 1 \vee \neg S_0(x, y) \vee S_1(x', 0)$  is part of a timed automaton with two clocks  $x$  and  $y$  modeled in BS(LRA). It represents

a transition from state  $S_0$  to state  $S_1$  that can be traversed only if clock  $y$  is at least 5 and that resets  $y$  to 0 and increases  $x$  by 1.

Arithmetic terms are constructed from a set  $\mathcal{X}$  of *variables*, the set of integer constants  $c \in \mathbb{Z}$ , and binary function symbols  $+$  and  $-$  (written infix). Additionally, we allow multiplication  $\cdot$  if one of the factors is an integer constant. Multiplication only serves us as syntactic sugar to abbreviate other arithmetic terms, e.g.,  $x + x + x$  is abbreviated to  $3 \cdot x$ . Atoms in BS(LRA) are either *first-order atoms* (e.g.,  $P(13, x)$ ) or *(linear) arithmetic atoms* (e.g.,  $x < 42$ ). Arithmetic atoms are denoted by  $\lambda$  and may use the predicates  $\leq, <, \neq, =, >, \geq$ , which are written infix and have the expected fixed interpretation. We use  $\triangleleft$  as a placeholder for any of these predicates. Predicates used in first-order atoms are called *free*. *First-order literals* and related notation is defined as before. *Arithmetic literals* coincide with arithmetic atoms, since the arithmetic predicates are closed under negation, e.g.,  $\neg(x \geq 42) \equiv x < 42$ .

BS(LRA) clauses are defined as for BS but using BS(LRA) atoms. We often write clauses in the form  $A \parallel C$  where  $C$  is a clause solely built of free first-order literals and  $A$  is a multiset of LRA atoms called the *constraint* of the clause. A clause of the form  $A \parallel C$  is therefore also called a *constrained clause*. The semantics of  $A \parallel C$  is as follows:

$$A \parallel C \text{ iff } \left( \bigwedge_{\lambda \in A} \lambda \right) \rightarrow C \text{ iff } \left( \bigvee_{\lambda \in A} \neg \lambda \right) \vee C$$

For example, the clause  $x > 1 \vee y \neq 5 \vee \neg Q(x) \vee R(x, y)$  is also written  $x \leq 1, y = 5 \parallel \neg Q(x) \vee R(x, y)$ . The negation  $\neg(A \parallel C)$  of a constrained clause  $A \parallel C$  where  $C = A_1 \vee \dots \vee A_n \vee \neg B_1 \vee \dots \vee \neg B_m$  is thus equivalent to  $(\bigwedge_{\lambda \in A} \lambda) \wedge \neg A_1 \wedge \dots \wedge \neg A_n \wedge B_1 \wedge \dots \wedge B_m$ . Note that since the neutral element of conjunction is  $\top$ , an empty constraint is thus valid, i.e. equivalent to true.

An *assignment* for a constraint  $A$  is a substitution (denoted  $\beta$ ) that maps all variables in  $\text{vars}(A)$  to real numbers  $c \in \mathbb{R}$ . An assignment is a *solution* for a constraint  $A$  if all atoms  $\lambda \in A$  evaluate to true. A constraint  $A$  is *satisfiable* if there exists a solution for  $A$ . Otherwise it is *unsatisfiable*. Note that assignments can be extended to  $C$  by also mapping variables of the first-order sort accordingly.

A clause or clause set is *abstracted* if its first-order literals contain only variables or first-order constants. Every clause  $C$  is equivalent to an abstracted clause that is obtained by replacing each non-variable arithmetic term  $t$  that occurs in a first-order atom by a fresh variable  $x$  while adding an arithmetic atom  $x \neq t$  to  $C$ . We assume abstracted clauses for theory development, but we prefer non-abstracted clauses in examples for readability, e.g., a unit clause  $P(3, 5)$  is considered in the development of the theory as the clause  $x = 3, y = 5 \parallel P(x, y)$ . In the implementation, we mostly prefer abstracted clauses except that we allow integer constants  $c \in \mathbb{Z}$  to appear as arguments of first-order literals. In some cases, this makes it easier to recognize whether two clauses can be matched or not. For instance, we see by syntactic comparison that the two unit clauses  $P(3, 5)$  and  $P(0, 1)$  have no substitution  $\sigma$  such that  $P(3, 5) = P(0, 1)\sigma$ . For the abstracted

versions on the other hand,  $x = 3, y = 5 \parallel P(x, y)$  and  $u = 0, v = 1 \parallel P(u, v)$  we can find a matching substitution for the first-order part  $\sigma := \{u \mapsto x, v \mapsto y\}$  and would have to check the constraints semantically to exclude the matching.

*Hierarchical Resolution.* One inference rule, foundational to most algorithms for solving constrained first-order clauses, is *hierarchical resolution* [6]:

$$\frac{A_1 \parallel L_1 \vee C_1 \quad A_2 \parallel L_2 \vee C_2 \quad \sigma = \text{mgu}(L_1, \text{comp}(L_2))}{(A_1, A_2 \parallel C_1 \vee C_2)\sigma}$$

The conclusion is called *hierarchical resolvent* (of the two clauses in the premise). A *refutation* is the sequence of resolution steps that produces a clause  $A \parallel \perp$  with  $\mathcal{A}^{\text{LRA}} \models A\delta$  for some grounding  $\delta$ . Hierarchic resolution is sound and refutationally complete for the BS(LRA) clauses considered here, since every set  $N$  of BS(LRA) clauses is *sufficiently complete* [6], because all constannts of the arithmetic sort are numbers. Hence *hierarchical resolution* is sound and refutationally complete for  $N$  [6,7]. *Hierarchical unit resolution* is a special case of hierarchic resolution, that only combines two clauses in case one of them is a unit clause. Hierarchic unit resolution is sound and complete for HBS(LRA) [6,7], but not even refutationally complete for BS(LRA).

Most algorithms for Bernays-Schönfinkel, first-order logic, and beyond utilize resolution. The SCL(T) calculus for HBS(LRA) uses hierarchic resolution in order to learn from the conflicts it encounters during its search. The hierarchic superposition calculus on the other hand derives new clauses via hierarchic resolution based on an ordering. The goal is to either derive the empty clause or a saturation of the clause set, i.e., a state from which no new clauses can be derived. Each of those algorithms must derive new clauses in order to progress, but their subroutines also get progressively slower as more clauses are derived. In order to increase efficiency, it is necessary to eliminate clauses that are obsolete. One measure that determines whether a clause is useful or not is *redundancy*.

*Redundancy.* In order to define redundancy for constrained clauses, we need an  $\mathcal{H}$ -order, i.e., a well-founded, total, strict ordering  $\prec$  on ground literals such that literals in the constraints (in our case arithmetic literals) are always smaller than first-order literals. Such an ordering can be lifted to constrained clauses and sets thereof by its respective multiset extension. Hence, we overload any such order  $\prec$  for literals, constrained clauses, and sets of constrained clause if the meaning is clear from the context. We define  $\preceq$  as the reflexive closure of  $\prec$  and  $N^{\preceq A \parallel C} := \{D \mid D \in N \text{ and } D \preceq A \parallel C\}$ . An instance of an LPO [15] with appropriate precedence can serve as an  $\mathcal{H}$ -order.

**Definition 2 (Clause Redundancy).** A ground clause  $A \parallel C$  is redundant with respect to a set  $N$  of ground clauses and an  $\mathcal{H}$ -order  $\prec$  if  $N^{\preceq A \parallel C} \models A \parallel C$ . A clause  $A \parallel C$  is redundant with respect to a clause set  $N$  and an  $\mathcal{H}$ -order  $\prec$  if for all  $A' \parallel C' \in \text{gnd}(A \parallel C)$  the clause  $A' \parallel C'$  is redundant with respect to  $\text{gnd}(N)$ .

If a clause  $A \parallel C$  is redundant with respect to a clause set  $N$ , then it can be removed from  $N$  without changing its semantics. Determining clause redundancy is an undecidable problem [11, 63]. However, there are special cases of redundant clauses that can be easily checked, e.g., tautologies and subsumed clauses. Techniques for tautology deletion and subsumption deletion are the most common elimination techniques in modern first-order provers.

A *tautology* is a clause that evaluates to true independent of the predicate interpretation or assignment. It is therefore redundant with respect to all orders and clause sets; even the empty set.

**Corollary 3 (Tautology for Constrained Clauses).** *A clause  $A \parallel C$  is a tautology if the existential closure of  $\neg(A \parallel C)$  is unsatisfiable.*

Since  $\neg(A \parallel C)$  is essentially ground (by existential closure and skolemization), it can be solved with an appropriate SMT solver, i.e., an SMT solver that supports unquantified uninterpreted functions coupled with linear real arithmetic. In [2], it is recommended to check only the following conditions for tautology deletion in hierarchic superposition:

**Corollary 4 (Tautology Check).** *A clause  $A \parallel C$  is a tautology if the existential closure of  $A$  is unsatisfiable or if  $C$  contains two literals  $L_1$  and  $L_2$  with  $L_1 = \text{comp}(L_2)$ .*

The advantage is that the check on the first-order side of the clause is still purely syntactic and corresponds to the tautology check for pure first-order logic. Nonetheless, there are tautologies that are not captured by Corollary 4, e.g.,  $x = y \parallel P(x) \vee \neg P(y)$ . The SCL(T) calculus on the other hand requires no tautology checks because it never learns tautologies as part of its conflict analysis [1, 11, 21]. This property is also inherent to the propositional CDCL (Conflict Driven Clause Learning) approach [8, 34, 41, 55, 63].

### 3 Subsumption for Constrained Clauses

A *subsumed* constrained clause is a clause that is redundant with respect to a single clause in our clause set. Formally, subsumption is defined as follows.

**Definition 5. (Subsumption for Constrained Clauses [2]).** *A constrained clause  $A_1 \parallel C_1$  subsumes another constrained clause  $A_2 \parallel C_2$  if there exists a substitution  $\sigma$  such that  $C_1\sigma \subseteq C_2$ ,  $\text{vars}(A_1\sigma) \subseteq \text{vars}(A_2)$ , and the universal closure of  $A_2 \rightarrow (A_1\sigma)$  holds in LRA.*

Eliminating redundant clauses is crucial for the efficient operation of an automatic first-order theorem prover. Although subsumption is considered one of the easier redundancy relationships that we can check in practice, it is still a hard problem in general:

**Lemma 6. (Complexity of Subsumption in the BS Fragment).** *Deciding subsumption for a pair of BS clauses is NP-complete.*

*Proof.* Containment in NP follows from the fact that the size of subsumption matchers is limited by the subsumed clause and set inclusion of literals can be decided in polynomial time. For the hardness part, consider the following polynomial-time reduction from 3-SAT. Take a propositional clause set where all clauses have length three. Now introduce a 6-place predicate  $R$  and encode each propositional variable  $P$  by a first-order variable  $x_P$ . Then a propositional clause  $L_1 \vee L_2 \vee L_3$  can be encoded by an atom  $R(x_{P_1}, p_1, x_{P_2}, p_2, x_{P_3}, p_3)$  where  $p_i$  is 0 if  $L_i$  is negative and 1 otherwise and  $P_i$  is the predicate of  $L_i$ . This way the clause set  $N$  can be represented by a single BS clause  $C_N$ . Now construct a clause  $D$  that contains all atoms representing the way a clause of length three can become true by ground atoms over  $R$  and constants 0, 1. For example, it contains atoms like  $R(0, 0, \dots)$  and  $R(1, 1, \dots)$  representing that the first literal of a clause is true. Actually, for each such atom  $R(0, 0, \dots)$  the clause  $D$  contains  $|C_N|$  copies. Finally,  $C_N$  subsumes  $D$  if and only if  $N$  is satisfiable.  $\square$

In order to be efficient, modern theorem provers need to decide multiple thousand subsumption checks per second. In the pure first-order case, this is possible because of indexing and filtering techniques that quickly decide most subsumption checks [24, 25, 27–30, 33, 39, 40, 45–49, 52–54, 56, 59, 61, 62].

For BS(LRA) (and FOL(LRA)), there also exists research on how to perform the subsumption check in general [2, 36], but the literature contains no dedicated indexing or filtering techniques for the constraint part of the subsumption check. In this section and as the main contribution of this paper, we present the first such filtering techniques for BS(LRA). But first, we explain how to solve the subsumption check for constrained clauses in general.

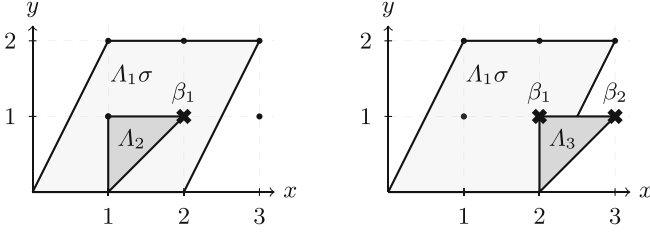
**First-Order Check.** The first step of the subsumption check is exactly the same as in first-order logic without arithmetic. We have to find a substitution  $\sigma$ , also called a *matcher*, such that  $C_1\sigma \subseteq C_2$ . The only difference is that it is not enough to compute one matcher  $\sigma$ , but we have to compute all matchers for  $C_1\sigma \subseteq C_2$  until we find one that satisfies the implication  $A_2 \rightarrow (A_1\sigma)$ . For instance, there are two matchers for the clauses  $C_1 := x + y \geq 0 \parallel Q(x, y)$  and  $C_2 := x < 0, y \geq 0 \parallel Q(x, x) \vee Q(y, y)$ . The matcher  $\{x \mapsto y\}$  satisfies the implication  $A_2 \rightarrow (A_1\sigma)$  and  $\{y \mapsto x\}$  does not. Our own algorithm for finding matchers is in the style of Stillman except that we continue after we find the first matcher [27, 58].

**Implication Check.** The universal closure of the implication  $A_2 \rightarrow (A_1\sigma)$  can be solved by any SMT solver for the respective theory after we negate it. Note that the resulting formula

$$\exists x_1, \dots, x_n. A_2 \wedge \neg(A_1\sigma) \quad \text{where } \{x_1, \dots, x_n\} = \text{vars}(A_2) \quad (1)$$

is already in clause normal form and that the formula can be treated as ground since existential variables can be handled as constants. Intuitively, the universal closure  $A_2 \rightarrow (A_1\sigma)$  asserts that the set of solutions satisfying  $A_2$  is a subset of





**Fig. 1.** Solutions of the constraints  $\Lambda_1\sigma$ ,  $\Lambda_2$ , and  $\Lambda_3$  depicted as polytopes

the set of solutions satisfying  $\Lambda_1\sigma$ . This means a solution to its negation (1) is a solution for  $\Lambda_2$ , but not for  $\Lambda_1\sigma$ , thus a counterexample of the subset relation.

*Example 7.* Let us now look at an example to illustrate the role that formula (1) plays in deciding subsumption. In our example, we have three clauses:  $\Lambda_1 \parallel C_1$ ,  $\Lambda_2 \parallel C_2$ , and  $\Lambda_3 \parallel C_2$ , where  $C_1 := \neg P(x, y) \vee Q(u, z)$ ,  $C_2 := \neg P(x, y) \vee Q(2, x)$ ,  $\Lambda_1 := y \geq 0, y \leq u, y \leq x + z, y \geq x + z - 2 \cdot u$ ,  $\Lambda_2 := x \geq 1, y \leq 1, y \geq x - 1$ , and  $\Lambda_3 := x \geq 2, y \leq 1, y \geq x - 2$ . Our goal is to test whether  $\Lambda_1 \parallel C_1$  subsumes the other two clauses. As our first step, we try to find a substitution  $\sigma$  such that  $C_1\sigma \subseteq C_2$ . The most general substitution fulfilling this condition is  $\sigma := \{z \mapsto x, u \mapsto 2\}$ . Next, we check whether  $\Lambda_1\sigma$  is implied by  $\Lambda_2$  and  $\Lambda_3$ . Normally, we would do so by solving the formula (1) with an SMT solver, but to help our intuitive understanding, we instead look at their solution sets depicted in Fig. 1. Note that  $\Lambda_1\sigma$  simplifies to  $\Lambda_1\sigma := y \geq 0, y \leq 2, y \leq 2 \cdot x, y \geq 2 \cdot x - 4$ . Here we see that the solution set for  $\Lambda_2$  is a subset of  $\Lambda_1\sigma$ . Hence,  $\Lambda_2$  implies  $\Lambda_1\sigma$ , which means that  $\Lambda_2 \parallel C_2$  is subsumed by  $\Lambda_1 \parallel C_1$ . The solution set for  $\Lambda_3$  is not a subset of  $\Lambda_1\sigma$ . For instance, the assignment  $\beta_2 := \{x \mapsto 3, y \mapsto 1\}$  is a counterexample and therefore a solution to the respective instance of formula (1). Hence,  $\Lambda_1 \parallel C_1$  does not subsume  $\Lambda_3 \parallel C_2$ .

**Excess Variables.** Note that in general it is not sufficient to find a substitution  $\sigma$  that matches the first-order parts to also match the theory constraints:  $C_1\sigma \subseteq C_2$  does not generally imply  $\text{vars}(\Lambda_1\sigma) \subseteq \text{vars}(\Lambda_2)$ . In particular, if  $\Lambda_1$  contains variables that do not appear in the first-order part  $C_1$ , then these must be projected to  $\Lambda_2$ . We arrive at a variant of (1), that is  $\exists x_1, \dots, x_n \forall y_1, \dots, y_m. \Lambda_2 \wedge \neg(\Lambda_1\sigma)$  where  $\{x_1, \dots, x_n\} = \text{vars}(\Lambda_2)$  and  $\{y_1, \dots, y_m\} = \text{vars}(\Lambda_1) \setminus \text{vars}(C_1)$ . Our solution to this problem is to normalize all clauses  $\Lambda \parallel C$  by eliminating all *excess variables*  $\mathcal{Y} := \text{vars}(\Lambda) \setminus \text{vars}(C)$  such that  $\text{vars}(\Lambda) \subseteq \text{vars}(C)$  is guaranteed. For linear real arithmetic this is possible with quantifier elimination techniques, e.g., Fourier-Motzkin elimination (FME). Although these techniques typically cause the size of  $\Lambda$  to increase exponentially, they often behave well in practice. In fact, we get rid of almost all excess variables in our benchmark examples with simplification techniques based on Gaussian elimination with execution time linear in the number of LRA atoms. Given the precondition  $\mathcal{Y} = \emptyset$  achieved by such elimination techniques,

we can compute  $\sigma$  as matcher for the first-order parts and then directly use it for testing whether the universal closure of  $A_2 \rightarrow (A_1\sigma)$  holds. An alternative solution to the issue of excess variables has been proposed: In [2], the substitution  $\sigma$  is decomposed as  $\sigma = \delta\tau$ , where  $\delta$  is the first-order matcher and  $\tau$  is a *theory matcher*, i.e.  $\text{dom}(\tau) \subseteq \mathcal{Y}$  and  $\text{vars}(\text{codom}(\tau)) \subseteq \text{vars}(A_2)$ . Then, exploiting Farkas' lemma, the computation of  $\tau$  is reduced to testing the feasibility of a linear program (restricted to matchers that are affine transformations).

The reduction to solving a linear program offers polynomial worst-case complexity but in practice typically behaves worse than solving the variant with quantifier alternations using an SMT solver such as Z3 [36, 42].

**Filtering First-Order Literals.** Even though deciding implication of theory constraints is in practice more expensive than constructing a matcher and deciding inclusion of first-order literals, we still incorporate some lightweight filters for our evaluation. Inspired by Schulz [54] we choose three features, so that every feature  $f$  maps clauses to  $\mathbb{N}_0$ , and  $f(C_1) \leq f(C_2)$  is necessary for  $C_1\sigma \subseteq C_2$ .

The features are:  $|C^+|$ , the number of positive first-order literals in  $C$ ,  $|C^-|$ , the number of negative first-order literals in  $C$ , and  $|C|$ , the number of occurrences of constants in  $C$ .

**Sample Point Heuristic.** The majority of subsumption tests fail because we cannot find a fitting substitution for their first-order parts. In our experiments, between 66.5% and 99.9% of subsumption tests failed this way. This means our tool only has to check in less than 33.5% of the cases whether one theory constraint implies the other. Despite this, our tool spends more time on implication checks than on the first-order part of the subsumption tests without filtering on the constraint implication tests. The reason is that constraint implication tests are typically much more expensive than the first-order part of a subsumption test. For this reason, we developed the *sample point heuristic* that is much faster to execute than a full constraint implication test, but still filters out the majority of implications that do not hold (in our experiments between 93.8% and 100%).

The idea behind the sample point heuristic is straightforward. We store for each clause  $A \parallel C$  a sample solution  $\beta$  for its theory constraint  $A$ . Before we execute a full constraint implication test, we simply evaluate whether the sample solution  $\beta$  for  $A_2$  is also a solution for  $A_1\sigma$ . If this is not the case, then  $\beta$  is a solution for (1) and a counterexample for the implication. If  $\beta$  is a solution for  $A_1\sigma$ , then the heuristic returns unknown and we have to execute a full constraint implication test, i.e., solve the SMT problem (1).

Often it is possible to get our sample solutions for free. Theorem provers based on hierarchic superposition typically check for every new clause  $A \parallel C$  whether  $A$  is satisfiable in order to eliminate tautologies. This means we can already use this tautology check to compute and store a sample solution for every new clause without extra cost. We only need to pick a solver for the check that returns a solution as a certificate of satisfiability. Although the SCL(T) calculus never learns any tautologies, it is also possible to get a sample solution for free as part of its conflict analysis [11].

*Example 8.* We revisit Example 7 to illustrate the sample point heuristic. During the tautology check for  $A_2 \parallel C_2$  and  $A_3 \parallel C_2$ , we determined that  $\beta_1 := \{x \mapsto 2, y \mapsto 1\}$  is a sample solution for  $A_2$  and  $\beta_2 := \{x \mapsto 3, y \mapsto 1\}$  a sample solution for  $A_3$ . Since  $A_2$  implies  $A_1\sigma$ , all sample solutions for  $A_2$  automatically satisfy  $A_1\sigma$ . This is the reason why the sample point heuristic never filters out an implication that actually holds, i.e., it returns unknown when we test whether  $A_2$  implies  $A_1\sigma$ . The assignment  $\beta_2$  on the other hand does not satisfy  $A_1\sigma$ . Hence, the sample point heuristic correctly claims that  $A_3$  does not imply  $A_1\sigma$ . Note that we could also have chosen  $\beta_1$  as the sample point for  $A_3$ . In this case, the sample point heuristic would also return unknown for the implication  $A_3 \rightarrow A_1\sigma$  although the implication does not hold.

**Trivial Cases.** Subsumption tests become much easier if the constraint  $A_i$  of one of the participating clauses is empty. We use two heuristic filters to exploit this fact. We highlight them here because they already exclude some subsumption tests before we reach the sample point heuristic in our implementation.

The *empty conclusion heuristic* exploits that  $A_1$  is valid if  $A_1$  is empty. In this case, all implications  $A_2 \rightarrow (A_1\sigma)$  hold because  $A_1\sigma$  evaluates to true under any assignment. So by checking whether  $A_1 = \emptyset$ , we can quickly determine whether  $A_2 \rightarrow (A_1\sigma)$  holds for some pairs of clauses. Note that in contrast to the sample point heuristic, this heuristic is used to find valid implications.

The *empty premise test* exploits that  $A_2$  is valid if  $A_2$  is empty. In this case, an implication  $A_2 \rightarrow (A_1\sigma)$  may only hold if  $A_1\sigma$  simplifies to the empty set as well. This is the case because any inequality in the canonical form  $\sum_{i=1}^n a_i x_i < c$  either simplifies to true (because  $a_i = 0$  for all  $i = 1, \dots, n$  and  $0 < c$  holds) and can be removed from  $A_1\sigma$ , or the inequality eliminates at least one assignment as a solution for  $A_1\sigma$  [51]. So if  $A_2 = \emptyset$ , we check whether  $A_1\sigma$  simplifies to the empty set instead of solving the SMT problem (1).

**Pipeline.** We call our approach a *pipeline* since it combines multiple procedures, which we call *stages*, that vary in complexity and are independent in principle, for the overall aim of efficiently testing subsumption. Pairs of clauses that “make it through” all stages, are those for which the subsumption relation holds. The pipeline is designed with two goals in mind: (1) To reject as many pairs of clauses as early as possible, and (2) to move stages further towards the end of the pipeline the more expensive they are.

The pipeline consists of six stages, all of which are mentioned above. We divide the pipeline into two phases, the *first-order phase* (FO-phase) consisting of two stages, and the *constraint phase* (C-phase), consisting of four stages. First-order filtering rejects all pairs of clauses for which  $f(C_1) > f(C_2)$  holds. Then, matching constructs all matchers  $\sigma$  such that  $C_1\sigma \subseteq C_2$ . Every matcher is individually tested in the constraint phase. Technically, this means that the input of all following stages is not just a pair of clauses, but a triple of two clauses and a matcher. The constraint phase then proceeds with the empty conclusion heuristic and the empty premise test to accept (resp. reject) all trivial cases of

**Algorithm 1:** Saturation prover used for evaluation

---

**Input** : A set  $N$  of clauses.  
**Output** :  $\perp$  or “unknown”.  
1  $U := \{C \in N \mid |C| = 1\}$   
2 **while**  $U \neq \emptyset$  **do**  
3      $M := \emptyset$   
4     **foreach**  $C \in U$  **do**  $M := M \cup \text{resolvents}(C, N)$   
5     **if**  $\perp \in M$  **then return**  $\perp$   
6     reduce  $M$  using  $N$  (forward subsumption)  
7     **if**  $M = \emptyset$  **then return** “unknown”  
8     reduce  $N$  using  $M$  (backward subsumption)  
9      $U := \{C \in M \mid |C| = 1\}$   
10     $N := N \cup M$   
11 **end**  
12 **return** “unknown”

---

the constraint implication test. The next stage is the sample point heuristic. If the sample solution  $\beta_2$  for  $A_2$  is no solution for  $A_1$  (i.e.  $\not\models A_1\sigma\beta_2$ ), then the matcher  $\sigma$  is rejected. Otherwise (i.e.  $\models A_1\sigma\beta_2$ ), the implication test  $A_2 \rightarrow (A_1\sigma)$  is performed by solving the SMT problem (1) to produce the overall result of the pipeline and finally determine whether subsumption holds.

## 4 Experimentation

In order to evaluate our new approach on three benchmark instances, derived from BS(LRA) applications, all presented techniques and their combination in form of a pipeline were implemented in the theorem prover SPASS-SPL, a prototype for BS(LRA) reasoning.

Note that SPASS-SPL contains more than one approach for BS(LRA) reasoning, e.g., the Datalog hammer for HBS(LRA) reasoning [10]. These various modes of operation operate independently, and the desired mode is chosen via command-line option. The reasoning approach discussed here is the current default option. On the first-order side, SPASS-SPL consists of a simple saturation prover based on hierarchic unit resolution, see Algorithm 1. It resolves unit clauses with other clauses until either the empty clause is derived or no new clauses can be derived. Note that this procedure is only complete for Horn clauses. For arithmetic reasoning, SPASS-SPL relies on SPASS-SATT, our sound and complete CDCL(LA) solver for quantifier-free linear real and linear mixed/integer arithmetic [12]. SPASS-SATT implements a version of the dual simplex algorithm fine-tuned towards SMT solving [16]. In order to ensure soundness, SPASS-SATT represents all numbers with the help of the *arbitrary-precision arithmetic library* FLINT [31]. This means all calculations, including the implication test and the sample point heuristic, are always exact and thus free of numerical errors. The most relevant part of SPASS-SPL with regards to

**Table 1.** Overview of how many clause pairs advance in the pipeline (top to bottom)

All		1c		bakery, tad		All	
		1 244 819k		196 437k		1 441 256k	
FO	Filtering	61.21%		85.03%		64.45%	
	$f(C_1) \leq f(C_2)$	761 905k	61.2061%	167 025k	85.0274%	928 931k	64.4540%
	Matching	0.02%		39.83%		7.18%	
	$C_1 \sigma \subseteq C_2$	131k	0.0106%	66 531k	33.8694%	66 664k	4.6254%
C	Empty (pre./con.)	44.73%		100.00%		99.89%	
	$\not\models \Lambda_1 \sigma, \not\models \Lambda_2$	59k	0.0047%	66 531k	33.8694%	66 591k	4.6203%
	Sample point	59.28%		0.12%		0.18%	
	$\models \Lambda_1 \sigma \beta_2$	35k	0.0028%	82k	0.0416%	117k	0.0081%
	Implication	95.51%		100.00%		98.66%	
	Subsumes	33k	0.0027%	82k	0.0416%	115k	0.0080%

**Table 2.** An overview of the accuracy of non-perfect pipeline stages

Test	Specificity/Sensitivity			Pos./Neg. Predictive Value		
Instances	1c	bakery, tad	All	1c	bakery, tad	All
FO Filtering	0.38797	0.14979	0.35552	0.00013	0.00049	0.00020
FO Matching	0.99996	0.60196	0.92841	0.78456	0.00123	0.00275
Empty Conclusion	0.70973	0.00000	0.00103	0.54474	0.00123	0.00173
Sample Point	0.93864	1.00000	0.99998	0.95510	1.00000	0.98653

this paper is that it performs tautology and subsumption deletion to eliminate redundant clauses. As a preprocessing step, SPASS-SPL eliminates all tautologies from the set of input clauses. Similarly, the function `resolvents( $C, N$ )` (see Line 4 of Algorithm 1) filters out all newly derived clauses that are tautologies. Note that we also use these tautology checks to eliminate all excess variables and to store sample solutions for all remaining clauses. After each iteration of the algorithm, we also check for subsumed clauses. We first eliminate newly generated clauses by forward subsumption (see Line 6 of Algorithm 1), then use the remaining clauses for backward subsumption (see Line 8 of Algorithm 1).

*Benchmarks.* Our benchmarking instances come out of three different applications. (1.) A supervisor for an automobile lane change assistant, formulated in the Horn fragment of BS(LRA) [9, 10] (five instances, referred to as **1c** in aggregate). (2.) The formalization of reachability for non-deterministic timed automata, formulated in the non-Horn fragment of BS(LRA) [20] (one instance, referred to as **tad**). (3.) Formalizations of variants of mutual exclusion protocols, such as the bakery protocol [38], also formulated in the non-Horn fragment of BS(LRA) [19] (one instance, referred to as **bakery**). The machine used for benchmarking features an Intel Xeon W-1290P CPU (10 cores, 20 threads, up to 5.2 GHz) and 64 GiB DDR4-2933 ECC main memory. Runtime was limited to ten minutes, and memory usage was not limited.

**Table 3.** Evaluation of the sample point heuristic

Instances		1c	bakery, tad	All
Bottleneck	(C time $\div$ FO time)			
	<i>without</i> sample point	127	2757	14867
	<i>with</i> sample point	78	32	89
Avg. pipeline runtime in $\mu s$				
	<i>without</i> sample point	0.0315	89.9401	0.5189
	<i>with</i> sample point	0.0311	1.4150	0.2197
Speedup	(C time <i>with</i> $\div$ <i>without</i> )	1.63	137.88	124.16
Benefit-to-cost	(C time <i>taken</i> $\div$ <i>saved</i> )	6.74	181.72	163.72

*Evaluation.* In Table 1 we give an overview of how many pairs of clauses advance how far in the pipeline (in thousands). Rows with grey background refer to a stage of the pipeline and show which portion of pairs of clauses were kept, relative to the previous stage. Rows with white background refer to (virtual) sets of clauses, their absolute size, and their size relative to the number of attempted tests, as well as the condition(s) established. The three groups of columns refer to groups of benchmark instances. Results vary greatly between 1c and the aggregate of bakery and tad. In 1c the relative number of subsumed clauses is significantly smaller (0.0027% compared to 0.0416%). FO Matching eliminates a large number of pairs in 1c, because the number of predicate symbols, and their arity (1c1, ..., 1c4: 36 predicates, arities up to 5; 1c5: 53 predicates, arities up to 12) is greater than in bakery (11 predicates, all of arity 2) and tad (4 predicates, all of arity 2).

*Binary Classifiers.* To evaluate the performance of each stage of the proposed test pipeline, we view each stage individually as a binary classifier on pairs of constrained clauses. The two classes we consider are “subsumes” (positive outcome) and “does not subsume” (negative outcome). Each stage of the pipeline computes a *prediction* on the *actual* result of the overall pipeline. We are thus interested in minimizing two kinds of errors: (1) When one stage of the pipeline predicts that the subsumption test will succeed (the prediction is positive) but it fails (the actual result is negative), called *false positive* (FP). (2) When one stage of the pipeline predicts that the subsumption test will fail (the prediction is negative) but it succeeds (the actual result is positive), called *false negative* (FN). Dually, a correct prediction is called *true positive* (TP) and *true negative* (TN). For each stage, at least one kind of error is excluded by design: First-order filtering and the sample point heuristic never produce false negatives. The empty conclusion heuristic never produces false positives. The empty premise test is perfect, i.e. it neither produces false positives nor false negatives, with the caveat of not always being applicable. The last stage (implication test) decides the overall result of the pipeline, and thus is also perfect. For evaluation of binary classifiers, we use four different measures (two symmetric pairs):

$$\text{SPC} = \text{TN} \div (\text{TN} + \text{FP}) \qquad \text{PPV} = \text{TP} \div (\text{TP} + \text{FP}) \qquad (2)$$

The first pair, *specificity* (SPC) and *positive predictive value*, see (2), is relevant only in presence of false positives (the measures approach 1 as FP approaches 0).

$$\text{SEN} = \text{TP} \div (\text{TP} + \text{FN}) \qquad \text{NPV} = \text{TN} \div (\text{TN} + \text{FN}) \qquad (3)$$

The second pair, *sensitivity* (SEN) and *negative predictive value* (NPV), see (3), is relevant only in presence of false negatives (the measures approach 1 as FN approaches 0). Specificity (resp. sensitivity) might be considered the “success rate” in our setup. They answer the question: “Given the *actual* result of the pipeline is ‘subsumed’ (resp. ‘not subsumed’), in how many cases does this stage *predict* correctly?” A specificity (resp. sensitivity) of 0.99 means that the classifier produces a false positive (resp. negative), i.e. a wrong prediction, in one out of one hundred cases. Both measures are independent of the prevalence of particular actual results, i.e. the measures are not biased by instances that feature many (or few) subsumed clauses. On the other hand, positive and negative predictive value are biased by prevalence. They answer the following question: “Given this stage of the pipeline *predicts* ‘subsumed’ (resp. ‘not subsumed’), how likely is it that the *actual* result indeed is ‘subsumed’ (resp. ‘not subsumed’)?”

In Table 2 we present for all non-perfect stages of the pipeline specificity (for those that produce false positives) and sensitivity (for those that produce false negatives) as well as the (positive/negative) predictive value. Note that the sample point heuristic has an exceptionally high specificity, still above 93% in the benchmarks where it performed worst. For the benchmarks **bakery** and **tad** it even performs perfectly. Combined, this gives a specificity of above 99.99%. Considering FO Filtering, we expect limited performance, since the structure of terms in BS is flat compared to the rich structure of terms as trees in full first-order logic. This is evidenced by a comparatively low specificity of 35%. However, this classifier is very easy to compute, so pays for itself. FO Matching is a much better classifier, at an aggregate sensitivity of 93%. Even though this classifier is NP-complete, this is not problematic in practice.

*Runtime.* In Table 3 we focus on the runtime improvement achieved by the sample point heuristic. In the first two lines (Bottleneck), we highlight how much slower testing implication of constraints (the C-phase) is compared to treating the first-order part (the FO-phase). This is equivalent to the time taken for the C-phase per pair of clauses (that reach at least the first C-phase) divided by the time taken for the FO-phase per pair of clauses. We see that without the sample point heuristic, we can expect the constraint implication test to take hundreds to thousands of times longer than the FO-phase. Adding the sample point heuristic decreases this ratio to below one hundred. In the fourth line (avg. pipeline runtime) we do not give a ratio, but the average time it takes to compute the whole pipeline. We achieve millions of subsumption checks per second. In the fifth line (Speedup), we take the time that all C-phases combined take per pair of clauses that reach at least the first C-phase, and take the ratio to the same time without applying the sample point heuristic. In the sixth line (Benefit-to-cost), we consider the time taken to compute the sample point vs. the time it saves. The benefit is about two orders of magnitude greater than the cost.

## 5 Conclusion

Our next step will be the integration of the subsumption test in the backward subsumption procedure of an SCL based reasoning procedure for BS(LRA) [11] which is currently under development.

There are various ways to improve the sample point heuristic. One improvement would be to store and check multiple sample points per clause. For instance, whenever the sample point heuristic fails and the implication test for  $\Delta_2 \rightarrow (\Delta_1 \sigma)$  also fails, store the solution to (1) as an additional sample point for  $\Delta_2$ . The new sample point will filter out any future implication tests with  $\Delta_1 \sigma$  or similar constraints. However, testing too many sample points might lead to costs outweighing benefits. A potential solution to this problem would be score-based garbage collection, as done in SAT solvers [57]. Another way to store and check multiple sample points per clause is to store a compact description of a set of points that is easy to check against. For instance, we can store the center point and edge length of the largest orthogonal hypercube contained in the solutions of a constraint, which is equivalent to infinitely many sample points. Computing the largest orthogonal hypercube for an LRA constraint is not much harder than finding a sample solution [14]. Checking whether a cube is contained in an LRA constraint works almost the same as evaluating a sample point [14].

Although we developed our sample point technique for the BS(LRA) fragment it is obvious that it will also work for the overall FOL(LRA) clause fragment, because this extension does not affect the LRA constraint part of clauses. From an automated reasoning perspective, satisfiability of the FOL(LRA) and BS(LRA) fragments (clause sets) is undecidable in both cases. Actually, satisfiability of a BS(LRA) clause set is already undecidable if the first-order part is restricted to a single monadic predicate [32]. The first-order part of BS(LRA) is decidable and therefore enables effective guidance for an overall reasoning procedure [11]. From an application perspective, the BS(LRA) fragment already encompasses a number of used (sub)languages. For example, timed automata [3] and a number of extensions thereof are contained in the BS(LRA) fragment [60].

We also believe that the sample point heuristic will speed up the constraint implication test for FOL(LIA), first-order clauses over linear integer arithmetic, FOL(NRA), i.e., first-order clauses over non-linear real arithmetic, and other combinations of FOL with arithmetic theories. However, the non-linear case will require a more sophisticated setup due to the nature of test points in this case, e.g., a solution may contain root expressions.

**Acknowledgments.** This work was partly funded by DFG grant 389792660 as part of TRR 248, see <https://perspicuous-computing.science>. We thank the anonymous reviewers for their thorough reading and detailed constructive comments. Martin Desharnais suggested some textual improvements.

## References

1. Alagi, G., Weidenbach, C.: NRCL - a model building approach to the Bernays-Schönfinkel fragment. In: Lutz, C., Ranise, S. (eds.) FroCoS 2015. LNCS (LNAI),



- vol. 9322, pp. 69–84. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-24246-0\\_5](https://doi.org/10.1007/978-3-319-24246-0_5)
2. Althaus, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic SUP(LA). In: Ghilardi, S., Sebastiani, R. (eds.) FroCoS 2009. LNCS (LNAI), vol. 5749, pp. 84–99. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-04222-5\\_5](https://doi.org/10.1007/978-3-642-04222-5_5)
  3. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**(2), 183–235 (1994). [https://doi.org/10.1016/0304-3975\(94\)90010-8](https://doi.org/10.1016/0304-3975(94)90010-8)
  4. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.* **4**(3), 217–247 (1994). <https://doi.org/10.1093/logcom/4.3.217>
  5. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) *Handbook of Automated Reasoning* (in 2 volumes), pp. 19–99. Elsevier and MIT Press, Cambridge (2001). <https://doi.org/10.1016/b978-044450813-3/50004-7>
  6. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. *Appl. Algebra Eng. Commun. Comput.* **5**, 193–212 (1994). <https://doi.org/10.1007/BF01190829>
  7. Baumgartner, P., Waldmann, U.: Hierarchic superposition revisited. In: Lutz, C., Sattler, U., Tinelli, C., Turhan, A.-Y., Wolter, F. (eds.) *Description Logic, Theory Combination, and All That*. LNCS, vol. 11560, pp. 15–56. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-22102-7\\_2](https://doi.org/10.1007/978-3-030-22102-7_2)
  8. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press, Amsterdam (2009)
  9. Bromberger, M., et al.: A sorted datalog hammer for supervisor verification conditions modulo simple linear arithmetic. CoRR abs/2201.09769 (2022). <https://arxiv.org/abs/2201.09769>
  10. Bromberger, M., Dragoste, I., Faqeh, R., Fetzer, C., Krötzsch, M., Weidenbach, C.: A datalog hammer for supervisor verification conditions modulo simple linear arithmetic. In: Konev, B., Reger, G. (eds.) FroCoS 2021. LNCS (LNAI), vol. 12941, pp. 3–24. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-86205-3\\_1](https://doi.org/10.1007/978-3-030-86205-3_1)
  11. Bromberger, M., Fiori, A., Weidenbach, C.: Deciding the Bernays-Schoenfinkel Fragment over bounded difference constraints by simple clause learning over theories. In: Henglein, F., Shoham, S., Vize, Y. (eds.) VMCAI 2021. LNCS, vol. 12597, pp. 511–533. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67067-2\\_23](https://doi.org/10.1007/978-3-030-67067-2_23)
  12. Bromberger, M., Fleury, M., Schwarz, S., Weidenbach, C.: SPASS-SATT. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 111–122. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-29436-6\\_7](https://doi.org/10.1007/978-3-030-29436-6_7)
  13. Bromberger, M., Leutgeb, L., Weidenbach, C.: An Efficient subsumption test pipeline for BS(LRA) clauses (2022). <https://doi.org/10.5281/zenodo.6544456>. Supplementary Material
  14. Bromberger, M., Weidenbach, C.: Fast cube tests for LIA constraint solving. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 116–132. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-40229-1\\_9](https://doi.org/10.1007/978-3-319-40229-1_9)
  15. Dershowitz, N.: Orderings for term-rewriting systems. *Theor. Comput. Sci.* **17**, 279–301 (1982). [https://doi.org/10.1016/0304-3975\(82\)90026-3](https://doi.org/10.1016/0304-3975(82)90026-3)
  16. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006). [https://doi.org/10.1007/11817963\\_11](https://doi.org/10.1007/11817963_11)

17. Eggers, A., Kruglov, E., Kupferschmid, S., Scheibler, K., Teige, T., Weidenbach, C.: Superposition Modulo Non-linear Arithmetic. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS (LNAI), vol. 6989, pp. 119–134. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-24364-6\\_9](https://doi.org/10.1007/978-3-642-24364-6_9)
18. Faqeh, R., Fetzter, C., Hermanns, H., Hoffmann, J., Klauck, M., Köhl, M.A., Steinmetz, M., Weidenbach, C.: towards dynamic dependable systems through evidence-based continuous certification. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. LNCS, vol. 12477, pp. 416–439. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-61470-6\\_25](https://doi.org/10.1007/978-3-030-61470-6_25)
19. Fietzke, A.: Labelled superposition. Ph.D. thesis, Universität des Saarlandes (2014). <https://doi.org/10.22028/D291-26569>
20. Fietzke, A., Weidenbach, C.: Superposition as a decision procedure for timed automata. *Math. Comput. Sci.* **6**(4), 409–425 (2012). <https://doi.org/10.1007/s11786-012-0134-5>
21. Fiori, A., Weidenbach, C.: SCL clause learning from simple models. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 233–249. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-29436-6\\_14](https://doi.org/10.1007/978-3-030-29436-6_14)
22. Fiori, A., Weidenbach, C.: SCL with theory constraints. CoRR abs/2003.04627 (2020). <https://arxiv.org/abs/2003.04627>
23. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *J. Satisf. Boolean Model. Comput.* **1**(3–4), 209–236 (2007). <https://doi.org/10.3233/sat190012>
24. Ganzinger, H., Nieuwenhuis, R., Nivela, P.: Fast term indexing with coded context trees. *J. Autom. Reason.* **32**(2), 103–120 (2004). <https://doi.org/10.1023/B:JARS.0000029963.64213.ac>
25. Gleiss, B., Kovács, L., Rath, J.: Subsumption demodulation in first-order theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 297–315. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51074-9\\_17](https://doi.org/10.1007/978-3-030-51074-9_17)
26. Gottlob, G.: Subsumption and implication. *Inf. Process. Lett.* **24**(2), 109–111 (1987). [https://doi.org/10.1016/0020-0190\(87\)90103-7](https://doi.org/10.1016/0020-0190(87)90103-7)
27. Gottlob, G., Leitsch, A.: On the efficiency of subsumption algorithms. *J. ACM* **32**(2), 280–295 (1985). <https://doi.org/10.1145/3149.214118>
28. Graf, P.: Extended path-indexing. In: Bundy, A. (ed.) CADE 1994. LNCS, vol. 814, pp. 514–528. Springer, Heidelberg (1994). [https://doi.org/10.1007/3-540-58156-1\\_37](https://doi.org/10.1007/3-540-58156-1_37)
29. Graf, P.: Substitution tree indexing. In: Hsiang, J. (ed.) RTA 1995. LNCS, vol. 914, pp. 117–131. Springer, Heidelberg (1995). [https://doi.org/10.1007/3-540-59200-8\\_52](https://doi.org/10.1007/3-540-59200-8_52)
30. Graf, P. (ed.): Term Indexing. LNCS, vol. 1053. Springer, Heidelberg (1995). <https://doi.org/10.1007/3-540-61040-5>
31. Hart, W.B.: Fast library for number theory: an introduction. In: Fukuda, K., Hoeven, J., Joswig, M., Takayama, N. (eds.) ICMS 2010. LNCS, vol. 6327, pp. 88–91. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15582-6\\_18](https://doi.org/10.1007/978-3-642-15582-6_18)
32. Horbach, M., Voigt, M., Weidenbach, C.: The universal fragment of presburger arithmetic with unary uninterpreted predicates is undecidable. CoRR abs/1703.01212 (2017). <http://arxiv.org/abs/1703.01212>
33. Purdom, P.W., Brown, C.A.: Fast many-to-one matching algorithms. In: Jouanaud, J.-P. (ed.) RTA 1985. LNCS, vol. 202, pp. 407–416. Springer, Heidelberg (1985). [https://doi.org/10.1007/3-540-15976-2\\_21](https://doi.org/10.1007/3-540-15976-2_21)

34. Bayardo, R.J., Schrag, R.: Using CSP look-back techniques to solve exceptionally hard SAT instances. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 46–60. Springer, Heidelberg (1996). [https://doi.org/10.1007/3-540-61551-2\\_65](https://doi.org/10.1007/3-540-61551-2_65)
35. Korovin, K., Voronkov, A.: Integrating Linear Arithmetic into Superposition Calculus. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 223–237. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-74915-8\\_19](https://doi.org/10.1007/978-3-540-74915-8_19)
36. Kruglov, E.: Superposition modulo theory. Ph.D. thesis, Universität des Saarlandes (2013). <https://doi.org/10.22028/D291-26547>
37. Kruglov, E., Weidenbach, C.: Superposition decides the first-order logic fragment over ground theories. *Math. Comput. Sci.* **6**(4), 427–456 (2012). <https://doi.org/10.1007/s11786-012-0135-4>
38. Lamport, L.: A new solution of dijkstra’s concurrent programming problem. *Commun. ACM* **17**(8), 453–455 (1974). <https://doi.org/10.1145/361082.361093>
39. McCune, W.: Otter 2.0. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 663–664. Springer, Heidelberg (1990). [https://doi.org/10.1007/3-540-52885-7\\_131](https://doi.org/10.1007/3-540-52885-7_131)
40. McCune, W.: Experiments with discrimination-tree indexing and path indexing for term retrieval. *J. Autom. Reason.* **9**(2), 147–167 (1992). <https://doi.org/10.1007/BF00245458>
41. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, 18–22 June 2001, pp. 530–535. ACM (2001). <https://doi.org/10.1145/378239.379017>
42. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_24](https://doi.org/10.1007/978-3-540-78800-3_24)
43. Nieuwenhuis, R., Hillenbrand, T., Riazanov, A., Voronkov, A.: On the evaluation of indexing techniques for theorem proving. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 257–271. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-45744-5\\_19](https://doi.org/10.1007/3-540-45744-5_19)
44. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 371–443. Elsevier and MIT Press, Cambridge (2001). <https://doi.org/10.1016/b978-044450813-3/50009-6>
45. Ohlbach, H.J.: Abstraction tree indexing for terms. In: 9th European Conference on Artificial Intelligence, ECAI 1990, Stockholm, Sweden, pp. 479–484 (1990)
46. Overbeek, R.A., Lusk, E.L.: Data structures and control architecture for implementation of theorem-proving programs. In: Bibel, W., Kowalski, R. (eds.) CADE 1980. LNCS, vol. 87, pp. 232–249. Springer, Heidelberg (1980). [https://doi.org/10.1007/3-540-10009-1\\_19](https://doi.org/10.1007/3-540-10009-1_19)
47. Ramakrishnan, I.V., Sekar, R.C., Voronkov, A.: Term indexing. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 1853–1964. Elsevier and MIT Press, Cambridge (2001). <https://doi.org/10.1016/b978-044450813-3/50028-x>
48. Riazanov, A., Voronkov, A.: Partially adaptive code trees. In: Ojeda-Aciego, M., de Guzmán, I.P., Brewka, G., Moniz Pereira, L. (eds.) JELIA 2000. LNCS (LNAI), vol. 1919, pp. 209–223. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-40006-0\\_15](https://doi.org/10.1007/3-540-40006-0_15)
49. Riazanov, A., Voronkov, A.: Efficient instance retrieval with standard and relational path indexing. *Inf. Comput.* **199**(1–2), 228–252 (2005). <https://doi.org/10.1016/j.ic.2004.10.012>

50. Robinson, J.A.: A machine-oriented logic based on the resolution principle. *J. ACM*, **12**(1), 23–41 (1965). <https://doi.org/10.1145/321250.321253>, <http://doi.acm.org/10.1145/321250.321253>
51. Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience series in discrete mathematics and optimization, Wiley, Hoboken (1999)
52. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: Proceedings of the IJCAR-2004 Workshop on Empirically Successful First-Order Theorem Proving. Elsevier Science (2004)
53. Schulz, S.: Fingerprint Indexing for Paramodulation and Rewriting. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 477–483. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31365-3\\_37](https://doi.org/10.1007/978-3-642-31365-3_37)
54. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS (LNAI), vol. 7788, pp. 45–67. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-36675-8\\_3](https://doi.org/10.1007/978-3-642-36675-8_3)
55. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Rutenbar, R.A., Otten, R.H.J.M. (eds.) Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, 10–14 November 1996, pp. 220–227. IEEE Computer Society/ACM (1996). <https://doi.org/10.1109/ICCAD.1996.569607>
56. Socher, R.: A subsumption algorithm based on characteristic matrices. In: Lusk, E., Overbeek, R. (eds.) CADE 1988. LNCS, vol. 310, pp. 573–581. Springer, Heidelberg (1988). <https://doi.org/10.1007/BFb0012858>
57. Soos, M., Kulkarni, R., Meel, K.S.: CrystalBall: gazing in the black box of SAT solving. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 371–387. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-24258-9\\_26](https://doi.org/10.1007/978-3-030-24258-9_26)
58. Stillman, R.B.: The concept of weak substitution in theorem-proving. *J. ACM* **20**(4), 648–667 (1973). <https://doi.org/10.1145/321784.321792>
59. Tammet, T.: Towards efficient subsumption. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS, vol. 1421, pp. 427–441. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0054276>
60. Voigt, M.: Decidable  $\exists^*\forall^*$  first-order fragments of linear rational arithmetic with uninterpreted predicates. *J. Autom. Reason.* **65**(3), 357–423 (2020). <https://doi.org/10.1007/s10817-020-09567-8>
61. Voronkov, A.: The anatomy of vampire implementing bottom-up procedures with code trees. *J. Autom. Reason.* **15**(2), 237–265 (1995). <https://doi.org/10.1007/BF00881918>
62. Voronkov, A.: Algorithms, datastructures, and other issues in efficient automated deduction. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 13–28. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-45744-5\\_3](https://doi.org/10.1007/3-540-45744-5_3)
63. Weidenbach, C.: Automated reasoning building blocks. In: Meyer, R., Platzer, A., Wehrheim, H. (eds.) Correct System Design. LNCS, vol. 9360, pp. 172–188. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-23506-6\\_12](https://doi.org/10.1007/978-3-319-23506-6_12)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



# Ground Joinability and Connectedness in the Superposition Calculus

André Duarte<sup>(✉)</sup>  and Konstantin Korovin<sup>(✉)</sup> 

The University of Manchester, Manchester, UK  
{andre.duarte,konstantin.korovin}@manchester.ac.uk

**Abstract.** Problems in many theories axiomatised by unit equalities (UEQ), such as groups, loops, lattices, and other algebraic structures, are notoriously difficult for automated theorem provers to solve. Consequently, there has been considerable effort over decades in developing techniques to handle these theories, notably in the context of Knuth-Bendix completion and derivatives. The superposition calculus is a generalisation of completion to full first-order logic; however it does not carry over all the refinements that were developed for it, and is therefore not a strict generalisation. This means that (i) as of today, even state of the art provers for first-order logic based on the superposition calculus, while more general, are outperformed in UEQ by provers based on completion, and (ii) the sophisticated techniques developed for completion are not available in any problem which is not in UEQ. In particular, this includes key simplifications such as ground joinability, which have been known for more than 30 years. In fact, all previous completeness proofs for ground joinability rely on proof orderings and proof reductions, which are not easily extensible to general clauses together with redundancy elimination. In this paper we address this limitation and extend superposition with ground joinability, and show that under an adapted notion of redundancy, simplifications based on ground joinability preserve completeness. Another recently explored simplification in completion is connectedness. We extend this notion to “ground connectedness” and show superposition is complete with both connectedness and ground connectedness. We implemented ground joinability and connectedness in a theorem prover, iProver, the former using a novel algorithm which we also present in this paper, and evaluated over the TPTP library with encouraging results.

**Keywords:** Superposition · Ground joinability · Connectedness · Closure redundancy · First-order theorem proving

## 1 Introduction

Automated theorem provers based on equational completion [4], such as Waldmeister, MædMax or Twee [13, 21, 25], routinely outperform superposition-based provers on unit equality problems (UEQ) in competitions such as CASC [22], despite the fact that the superposition calculus was developed as a generalisation

of completion to full clausal first-order logic with equality [19]. One of the main ingredients for their good performance is the use of ground joinability criteria for the deletion of redundant equations [1], among other techniques. However, existing proofs of refutational completeness of deduction calculi wrt. these criteria are restricted to unit equalities and rely on proof orderings and proof reductions [1, 2, 4], which are not easily extensible to general clauses together with redundancy elimination.

Since completion provers perform very poorly (or not at all) on non-UEQ problems (relying at best on incomplete transformations to unit equality [8]), this motivates an attempt to transfer those techniques to the superposition calculus and prove their completeness, so as to combine the generality of the superposition calculus with the powerful simplification rules of completion. To our knowledge, no prover for first-order logic incorporates ground joinability redundancy criteria, except for particular theories such as associativity-commutativity (AC) [20].

For instance, if  $f(x, y) \approx f(y, x)$  is an axiom, then the equation  $f(x, f(y, z)) \approx f(x, f(z, y))$  is redundant, but this cannot be justified by any simplification rule in the superposition calculus. On the other hand, a completion prover which implements ground joinability can easily delete the latter equation wrt. the former. We show that ground joinability can be enabled in the superposition calculus without compromising completeness.

As another example, the simplification rule in completion can use  $f(x) \approx s$  (when  $f(x) \succ s$ ) to rewrite  $f(a) \approx t$  regardless of how  $s$  and  $t$  compare, while the corresponding demodulation rule in superposition can only rewrite if  $s \prec t$ . Our “encompassment demodulation” rule matches the former, while also being complete in the superposition calculus.

In [11] we introduced a novel theoretical framework for proving completeness of the superposition calculus, based on an extension of Bachmair-Ganzinger model construction [5], together with a new notion of redundancy called “closure redundancy”. We used it to prove that certain AC joinability criteria, long used in the context of completion [1], could also be incorporated in the superposition calculus for full first-order logic while preserving completeness.

In this paper, we extend this framework to show the completeness of the superposition calculus extended with: (i) a general ground joinability simplification rule, (ii) an improved encompassment demodulation simplification rule, (iii) a connectedness simplification rule extending [3, 21], and (iv) a new ground connectedness simplification rule. The proof of completeness that enables these extensions is based on a new encompassment closure ordering. In practice, these extensions help superposition to be competitive with completion in UEQ problems, and improves the performance on non-UEQ problems, which currently do not benefit from these techniques at all.

We also present a novel incremental algorithm to check ground joinability, which is very efficient in practice; this is important since ground joinability can be an expensive criterion to test. Finally, we discuss some of the experimental results we obtained after implementing these techniques in iProver [10, 16].

The paper is structured as follows. In Sect. 2 we define some basic notions to be used throughout the paper. In Sect. 3 we define the closure ordering we use to



prove redundancies. In Sect. 4 we present redundancy criteria for demodulation, ground joinability, connectedness, and ground connectedness. We prove their completeness in the superposition calculus, and discuss a concrete algorithm for checking ground joinability, and how it may improve on the algorithms used in e.g. Waldmeister [13] or Twee [21]. In Sect. 5 we discuss experimental results.

## 2 Preliminaries

We consider a signature consisting of a finite set of function symbols and the equality predicate as the only predicate symbol. We fix a countably infinite set of variables. First-order *terms* are defined in the usual manner. Terms without variables are called *ground* terms. A *literal* is an unordered pair of terms with either positive or negative polarity, written  $s \approx t$  and  $s \not\approx t$  respectively (we write  $s \approx t$  to mean either of the former two). A *clause* is a multiset of literals. Collectively terms, literals, and clauses will be called *expressions*.

A *substitution* is a mapping from variables to terms which is the identity for all but finitely many variables. An injective substitution onto variables is called a *renaming*. If  $e$  is an expression, we denote application of a substitution  $\sigma$  by  $e\sigma$ , replacing all variables with their image in  $\sigma$ . Let  $\text{GSubs}(e) = \{\sigma \mid e\sigma \text{ is ground}\}$  be the set of *ground substitutions* for  $e$ . Overloading this notation for sets we write  $\text{GSubs}(E) = \{\sigma \mid \forall e \in E. e\sigma \text{ is ground}\}$ . Finally, we write e.g.  $\text{GSubs}(e_1, e_2)$  instead of  $\text{GSubs}(\{e_1, e_2\})$ . The identity substitution is denoted by  $\epsilon$ .

A substitution  $\theta$  is *more general* than  $\sigma$  if  $\theta\rho = \sigma$  for some substitution  $\rho$  which is not a renaming. If  $s$  and  $t$  can be *unified*, that is, if there exists  $\sigma$  such that  $s\sigma = t\sigma$ , then there also exists the *most general unifier*, written  $\text{mgu}(s, t)$ . A term  $s$  is said to be *more general* than  $t$  if there exists a substitution  $\theta$  that makes  $s\theta = t$  but there is no substitution  $\sigma$  such that  $t\sigma = s$ . Two terms  $s$  and  $t$  are said to be *equal modulo renaming* if there exist injective  $\theta, \sigma$  such that  $s\theta = t$  and  $t\sigma = s$ . The relations “less general than”, “equal modulo renaming”, and their union are represented respectively by the symbols  $\sqsupset$ ,  $\equiv$ , and  $\sqcup$ .

A more refined notion of instance is that of *closure* [6]. Closures are pairs  $e \cdot \sigma$  that are said to *represent* the expression  $e\sigma$  while retaining information about the original term and its instantiation. Closures where  $e\sigma$  is ground are said to be *ground closures*. Let  $\text{GClos}(e) = \{e \cdot \sigma \mid e\sigma \text{ is ground}\}$  be the set of ground closures of  $e$ . Overloading the notation for sets, if  $N$  is a set of clauses then  $\text{GClos}(N) = \bigcup_{C \in N} \text{GClos}(C)$ .

We write  $s[t]$  if  $t$  is a *subterm* of  $s$ . If also  $s \neq t$ , then it is a *strict subterm*. We denote these relations by  $s \sqsupseteq t$  and  $s \supset t$  respectively. We write  $s[t \mapsto t']$  to denote the term obtained from  $s$  by replacing all occurrences of  $t$  by  $t'$ .

A (strict) partial order is a binary relation which is transitive ( $a \succ b \succ c \Rightarrow a \succ c$ ), irreflexive ( $a \not\succ a$ ), and asymmetric ( $a \succ b \Rightarrow b \not\succ a$ ). A (non-strict) partial preorder (or quasiorder) is any transitive, reflexive relation. A (pre)order is total over  $X$  if  $\forall x, y \in X. x \succeq y \vee y \succeq x$ . Whenever a non-strict (pre)order  $\succeq$  is given, the induced equivalence relation  $\sim$  is  $\succeq \cap \succeq$ , and the induced strict pre(order)  $\succ$  is  $\succeq \setminus \sim$ . The *transitive closure* of a relation  $\succ$ , the smallest transitive



relation that contains  $\succ$ , is denoted by  $\succ^+$ . A *transitive reduction* of a relation  $\succ$ , the smallest relation whose transitive closure is  $\succ$ , is denoted by  $\succ^-$ .

For an ordering  $\succ$  over a set  $X$ , its *multiset extension*  $\succ$  over multisets of  $X$  is given by:  $A \succ B$  iff  $A \neq B$  and  $\forall x \in B. B(x) > A(x) \exists y \in A. y \succ x \wedge A(y) > B(y)$ , where  $A(x)$  is the number of occurrences of element  $x$  in multiset  $A$  (we also use  $\succ$  for the the multiset extension of  $\succ$ ). It is well known that the multiset extension of a well-founded/total order is also a well-founded/total order, respectively [9]. The *(n-fold) lexicographic extension* of  $\succ$  over  $X$  is denoted  $\succ_{\text{lex}}$  over ordered  $n$ -tuples of  $X$ , and is given by  $\langle x_1, \dots, x_n \rangle \succ_{\text{lex}} \langle y_1, \dots, y_n \rangle$  iff  $\exists i. x_1 = y_1 \wedge \dots \wedge x_{i-1} = y_{i-1} \wedge x_i \succ y_i$ . The lexicographic extension of a well-founded/total order is also a well-founded/total order, respectively.

A binary relation  $\rightarrow$  over the set of terms is a *rewrite relation* if (i)  $l \rightarrow r \Rightarrow l\sigma \rightarrow r\sigma$  and (ii)  $l \rightarrow r \Rightarrow s[l] \rightarrow s[r]$ . The *reflexive-transitive closure* of a relation is the smallest reflexive-transitive relation which contains it. It is denoted by  $\rightarrow^*$ . Two terms are *joinable* ( $s \downarrow t$ ) if  $s \rightarrow^* u \leftarrow^* t$ .

If a rewrite relation is also a strict ordering, then it is a *rewrite ordering*. A *reduction ordering* is a rewrite ordering which is well-founded. In this paper we consider reduction orderings which are total on ground terms, such orderings are also *simplification orderings* i.e., satisfy  $s \triangleright t \Rightarrow s \succ t$ .

### 3 Ordering

In [11] we presented a novel proof of completeness of the superposition calculus based on the notion of closure redundancy, which enables the completeness of stronger redundancy criteria to be shown, including AC normalisation, AC joinability, and encompassment demodulation. In this paper we use a slightly different closure ordering ( $\succ_{cc}$ ), in order to extract better completeness conditions for the redundancy criteria that we present in this paper (the definition of closure redundant clause and closure redundant inference is parametrised by this  $\succ_{cc}$ ).

Let  $\succ_t$  be a simplification ordering which is total on ground terms. We extend this first to an ordering on ground term closures, then to an ordering on ground clause closures. Let

$$s \cdot \sigma \succ_{tc'} t \cdot \rho \quad \text{iff} \quad \begin{array}{l} \text{either } s\sigma \succ_t t\rho \\ \text{or else } s\sigma = t\rho \text{ and } s \sqsupset t, \end{array} \quad (1)$$

where  $s\sigma$  and  $t\rho$  are ground, and let  $\succ_{tc}$  be an (arbitrary) total well-founded extension of  $\succ_{tc'}$ . We extend this to an ordering on clause closures. First let

$$M_{lc}((s \approx t) \cdot \theta) = \{s\theta \cdot \epsilon, t\theta \cdot \epsilon\}, \quad (2)$$

$$M_{lc}((s \not\approx t) \cdot \theta) = \{s\theta \cdot \epsilon, t\theta \cdot \epsilon, s\theta \cdot \epsilon, t\theta \cdot \epsilon\}, \quad (3)$$

and let  $M_{cc}$  be defined as follows, depending on whether the clause is unit or non-unit:

$$M_{cc}(\emptyset \cdot \theta) = \emptyset, \quad (4)$$

$$M_{cc}((s \approx t) \cdot \theta) = \{\{s \cdot \theta\}, \{t \cdot \theta\}\}, \quad (5)$$

$$M_{cc}((s \not\approx t) \cdot \theta) = \{\{s \cdot \theta, t \cdot \theta, s\theta \cdot \epsilon, t\theta \cdot \epsilon\}\}, \quad (6)$$

$$M_{cc}((s \approx t \vee \dots) \cdot \theta) = \{M_{lc}(L \cdot \theta) \mid L \in (s \approx t \vee \dots)\}, \quad (7)$$

then  $\succ_{cc}$  is defined by

$$C \cdot \sigma \succ_{cc} D \cdot \rho \quad \text{iff} \quad M_{cc}(C \cdot \sigma) \gg_{tc} M_{cc}(D \cdot \rho). \quad (8)$$

The main purpose of this definition is twofold: (i) that when  $s\theta \succ_t t\theta$  and  $u$  occurs in a clause  $D$ , then  $s\theta \triangleleft u$  or  $s \sqsubset s\theta = u$  implies  $(s \approx t) \cdot \theta \rho \prec_{cc} D \cdot \rho$ , and (ii) that when  $C$  is a positive unit clause,  $D$  is not,  $s$  is the maximal subterm in  $C\theta$  and  $t$  is the maximal subterm in  $D\sigma$ , then  $s \succeq_t t$  implies  $C \cdot \theta \prec_{cc} D \cdot \sigma$ . These two properties enable unconditional rewrites via oriented unit equations on positive unit clauses to succeed whenever they would also succeed in unfailing completion [4], and rewrites on negative unit and non-unit clauses to always succeed. This will enable us to prove the correctness of the simplification rules presented in the following section.

## 4 Redundancies

In this section we present several redundancy criteria for the superposition calculus and prove their completeness. Recall the definitions in [11]: a clause  $C$  is redundant in a set  $S$  if all its ground closures  $C \cdot \theta$  follow from closures in  $\text{GClos}(S)$  which are smaller wrt.  $\succ_{cc}$ ; an inference  $C_1, \dots, C_n \vdash D$  is redundant in a set  $S$  if, for all  $\theta \in \text{GSubs}(C_1, \dots, C_n, D)$  such that  $C_1\theta, \dots, C_n\theta \vdash D\theta$  is a valid inference, the closure  $D \cdot \theta$  follows from closures in  $\text{GClos}(S)$  such that each is smaller than some  $C_1 \cdot \theta, \dots, C_n \cdot \theta$ . These definitions (in terms of ground closures rather than in terms of ground clauses, as in [19]) arise because they enable us to justify stronger redundancy criteria for application in superposition theorem provers, including the AC criteria developed in [11] and the criteria in this section.

**Theorem 1.** The superposition calculus [19] is refutationally complete wrt. closure redundancy, that is, if a set of clauses is saturated up to closure redundancy (meaning any inference with non-redundant premises in the set is redundant) and does not contain the empty clause, then it is satisfiable.

*Proof.* The proof of completeness of the superposition calculus wrt. this closure ordering carries over from [11] with some modifications, which are presented in a full version of this paper [12].

### 4.1 Encompassment Demodulation

We introduce the following definition, to be re-used throughout the paper.

**Definition 1.** A rewrite via  $l \approx r$  in clause  $C[l\theta]$  is *admissible* if one of the following conditions holds: (i)  $C$  is not a positive unit, or (let  $C = s[l\theta] \approx t$  for some  $\theta$ ) (ii)  $l\theta \neq s$ , or (iii)  $l\theta \sqsupset l$ , or (iv)  $s \prec_t t$ , or (v)  $r\theta \prec_t t$ .<sup>1</sup>

<sup>1</sup> We note that (iv) is superfluous, but we include it since in practice it is easier to check, as it is local to the clause being rewritten and therefore needs to be checked only once, while (v) needs to be checked with each demodulation attempt.

We then have

$$\begin{array}{l} \text{Encompassment} \\ \text{Demodulation} \end{array} \quad \frac{l \approx r \quad C[l\theta]}{C[l\theta \mapsto r\theta]}, \quad \begin{array}{l} \text{where } l\theta \succ_t r\theta, \text{ and} \\ \text{rewrite via } l \approx r \text{ in } C \text{ is admissible.} \end{array} \quad (9)$$

In other words, given an equation  $l \approx r$ , if an instance  $l\theta$  is a subterm in  $C$ , then the rewrite is admissible (meaning, for example, that an unconditional rewrite is allowed when  $l\theta \succ_t r\theta$ ) if  $C$  is not a positive unit, or if  $l\theta$  occurs at a strict subterm position, or if  $l\theta$  is less general than  $l$ , or if  $l\theta$  occurs outside a maximal side, or if  $r\theta$  is smaller than the other side. This restriction is much weaker than the one given for the usual demodulation rule in superposition [17], and equivalent to the one in equational completion when we restrict ourselves to unit equalities [4].

*Example 1.* If  $f(x) \succ_t s$ , we can use  $f(x) \approx s$  to rewrite  $f(x) \approx t$  when  $s \prec_t t$ , and  $f(a) \approx t$ ,  $f(x) \not\approx t$ , or  $f(x) \approx t \vee C$  regardless of how  $s$  and  $t$  compare.

## 4.2 General Ground Joinability

In [11] we developed redundancy criteria for the theory of AC functions in the superposition calculus. In this section we extend these techniques to develop redundancy criteria for ground joinability in arbitrary equational theories.

**Definition 2.** Two terms are *strongly joinable* ( $s \downarrow t$ ), in a clause  $C$  wrt. a set of equations  $S$ , if either  $s = t$ , or  $s \rightarrow s[l_1\sigma_1 \mapsto r_1\sigma_1] \xrightarrow{*} t$  via rules  $l_i \approx r_i \in S$ , where the rewrite via  $l_1 \approx r_1$  is admissible in  $C$ , or  $s \rightarrow s[l_1\sigma_1 \mapsto r_1\sigma_1] \downarrow t[l_2\sigma_2 \mapsto r_2\sigma_2] \leftarrow t$  via rules  $l_i \approx r_i \in S$ , where the rewrites via  $l_1 \approx r_1$  and  $l_2 \approx r_2$  are admissible in  $C$ . To make the ordering explicit, we may write  $s \downarrow_{\succ} t$ . Two terms are *strongly ground joinable* ( $s \Downarrow t$ ), in a clause  $C$  wrt. a set of equations  $S$ , if for all  $\theta \in \text{GSubs}(s, t)$  we have  $s\theta \downarrow t\theta$  in  $C$  wrt.  $S$ .

We then have:

$$\text{Ground joinability} \quad \frac{s \approx t \vee C \quad S}{S}, \quad \text{where } s \Downarrow t \text{ in } s \approx t \vee C \text{ wrt. } S, \quad (10a)$$

$$\text{Ground joinability} \quad \frac{s \not\approx t \vee C \quad S}{C}, \quad \text{where } s \Downarrow t \text{ in } s \not\approx t \vee C \text{ wrt. } S. \quad (10b)$$

**Theorem 2.** Ground joinability is a sound and admissible redundancy criterion of the superposition calculus wrt. closure redundancy.

*Proof.* We will show the positive case first. If  $s \Downarrow t$ , then for any instance  $(s \approx t \vee C) \cdot \theta$  we either have  $s\theta = t\theta$ , and therefore  $\emptyset \models (s \approx t) \cdot \theta$ , or we have wlog.  $s\theta \succ_t t\theta$ , with  $s\theta \downarrow t\theta$ . Then  $s\theta$  and  $t\theta$  can be rewritten to the same normal form  $u$  by  $l_i\sigma_i \rightarrow r_i\sigma_i$  where  $l_i \approx r_i \in S$ . Since  $u \prec_t s\theta$  and  $u \succeq_t t\theta$ , then  $(s \approx t \vee C) \cdot \theta$

follows from smaller  $(u \approx u \vee C) \cdot \theta^2$  (a tautology, i.e. follows from  $\emptyset$ ) and from the instances of clauses in  $S$  used to rewrite  $s\theta \rightarrow u \leftarrow t\theta$ . It only remains to show that these latter instances are also smaller than  $(s \approx t \vee C) \cdot \theta$ . Since we have assumed  $s\theta \succ_t t\theta$ , then at least one rewrite step must be done on  $s\theta$ . Let  $l_1\sigma_1 \rightarrow r_1\sigma_1$  be the instance of the rule used for that step, with  $(l_1 \approx r_1) \cdot \sigma_1$  the closure that generates it. By Definition 1 and 2, one of the following holds:

- $C \neq \emptyset$ , therefore  $(l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t \vee C) \cdot \theta$ , or
- $l_1\sigma_1 \triangleleft s\theta$ , therefore  $l_1\sigma_1 \prec_t s\theta \Rightarrow l_1 \cdot \sigma_1 \prec_{tc} s \cdot \theta \Rightarrow (l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t) \cdot \theta$ ,  
or
- $l_1\sigma_1 = s\theta$  and  $s \sqsupset l_1$ , therefore  $l_1 \cdot \sigma_1 \prec_{tc} s \cdot \theta \Rightarrow (l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t) \cdot \theta$ ,  
or
- $l_1\sigma_1 = s\theta$  and  $s \equiv l_1$  and  $r_1\sigma_1 \prec_t t\theta$ , therefore  $r_1 \cdot \sigma_1 \prec_{tc} t \cdot \theta \Rightarrow (l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t) \cdot \theta$ .

As for the remaining steps, they are done on the smaller side  $t\theta$  or on the other side after this first rewrite, which is smaller than  $s\theta$ . Therefore all subsequent steps done by any  $l_j\sigma_j \rightarrow r_j\sigma_j$  will have  $r_j \cdot \sigma_j \prec_{tc} l_j \cdot \sigma_j \prec_{tc} s \cdot \theta \Rightarrow (l_j \approx r_j) \cdot \sigma_j \prec_{cc} (s \approx t \vee C) \cdot \theta$ . As such, since this holds for all ground closures  $(s \approx t \vee C) \cdot \theta$ , then  $s \approx t \vee C$  is redundant wrt.  $S$ .

For the negative case, the proof is similar. We will conclude that  $(s \not\approx t \vee C) \cdot \theta$  follows from smaller  $(l_i \approx r_i) \cdot \sigma_i \in \text{GClos}(S)$  and smaller  $(u \not\approx u \vee C) \cdot \theta$ . The latter, of course, follows from smaller  $C \cdot \theta$ , therefore  $s \not\approx t \vee C$  is redundant wrt.  $S \cup \{C\}$ .  $\square$

*Example 2.* If  $S = \{f(x, y) \approx f(y, x)\}$ , then  $f(x, f(y, z)) \approx f(x, f(z, y))$  is redundant wrt.  $S$ . Note that  $f(x, y) \approx f(y, x)$  is not orientable by any simplification ordering, therefore this cannot be justified by demodulation alone.

**Testing for Ground Joinability.** The general criterion presented above begs the question of how to test, in practice, whether  $s \S t$  in a clause  $s \approx t \vee C$ . Several such algorithms have been proposed [1, 18, 21]. All of these are based on the observation that if we consider all total preorders  $\succeq_v$  on  $\text{Vars}(s, t)$  and for all of them show strong joinability with a modified ordering—which we denote  $\succ_{t[v]}$ —then we have shown strong *ground* joinability in the order  $\succ_t$  [18].

**Definition 3.** A simplification order on terms  $\succ_t$  *extended with* a preorder on variables  $\succeq_v$ , denoted  $\succeq_{t[v]}$ , is a simplification preorder (i.e. satisfies all the relevant properties in Sect. 2) such that  $\succeq_{t[v]} \supseteq \succ_t \cup \succeq_v$ .

*Example 3.* If  $x \succ_v y$ , then  $g(x) \succ_{t[v]} g(y)$ ,  $g(x) \succ_{t[v]} y$ ,  $f(x, y) \succ_{t[v]} f(y, x)$ , etc.

The simplest algorithm based on this approach would be to enumerate all possible total preorders  $\succeq_v$  over  $\text{Vars}(s, t)$ , and exhaustively reduce both sides

<sup>2</sup> Wlog.  $u\theta = u$ , renaming variables in  $u$  if necessary.

via equations in  $S$  orientable by  $\succ_{t[v]}$ , checking if the terms can be reduced to the same normal form for all total preorders. This is very inefficient since there are  $\mathcal{O}(n!e^n)$  such total preorders [7], where  $n$  is the cardinality of  $\text{Vars}(s, t)$ . Another approach is to consider only a smaller number of partial preorders, based on the obvious fact that  $s \not\prec_{\succ_{t[v]}} t \Rightarrow \forall \succeq'_v \supseteq \succeq_v. s \not\prec_{\succ_{t[v']}} t$ , so that joinability under a smaller number of partial preorders can imply joinability under all the total preorders, necessary to prove ground joinability.

However, this poses the question of how to choose which partial preorders to check. Intuitively, for performance, we would like that whenever the two terms are *not* ground joinable, that some total preorder where they are not joinable is found as early as possible, and that whenever the two terms *are* joinable, that all total preorders are covered in as few partial preorders as possible.

*Example 4.* Let  $S = \{f(x, f(y, z)) \approx f(y, f(x, z))\}$ . Then  $f(x, f(y, f(z, f(w, u)))) \approx f(x, f(y, f(w, f(z, u))))$  can be shown to be ground joinable wrt.  $S$  by checking just three cases:  $\succeq_v \in \{z \succ w, z \sim w, z \prec w\}$ , even though there are 6942 possible preorders.

Waldmeister first tries all partial preorders relating two variables among  $\text{Vars}(s, t)$ , then three, etc. until success, failure (by trying a total order and failing to join) or reaching a predefined limit of attempts [1]. Twee tries an arbitrary total strict order, then tries to weaken it, and repeats until all total preorders are covered [21]. We propose a novel algorithm—**incremental ground joinability**—whose main improvement is *guiding* the process of picking which preorders to check by finding, during the process of searching for rewrites on subterms of the terms we are attempting to join, minimal extensions of the term order with a variable preorder which allow the rewrite to be done in the  $\succ$  direction.

Our algorithm is summarised as follows. We start with an empty queue of variable preorders,  $V$ , initially containing only the empty preorder. Then, while  $V$  is not empty, we pop a preorder  $\succeq_v$  from the queue, and attempt to perform a rewrite via an equation which is newly orientable by some extension  $\succeq'_v$  of  $\succeq_v$ . That is, during the process of finding generalisations of a subterm of  $s$  or  $t$  among left-hand sides of candidate unoriented unit equations  $l \approx r$ , when we check that the instance  $l\theta \approx r\theta$  used to rewrite is oriented, we try to force this to be true under some minimal extension  $\succ_{t[v']}$  of  $\succ_{t[v]}$ , if possible. If no such rewrite exists, the two terms are not strongly joinable under  $\succ_{t[v]}$  or any extension, and so are not strongly ground joinable and we are done. If it exists, we exhaustively rewrite with  $\succ_{t[v']}$ , and check if we obtain the same normal form. If we do not obtain it yet, we repeat the process of searching rewrites via equations orientable by further extensions of the preorder. But if we do, then we have proven joinability in the extended preorder; now we must add back to the queue a set of preorders  $O$  such that all the total preorders which are  $\supseteq \succeq_v$  (popped from the queue) but not  $\supseteq \succeq'_v$  (minimal extension under which we have proven joinability) are  $\supseteq$  of some  $\succeq''_v \in O$  (pushed back into the queue to be checked). Obtaining this  $O$  is implemented by  $\text{order\_diff}(\succeq_v, \succeq'_v)$ , defined below. Whenever there are no more preorders in the queue to check, then we have checked that the terms are strongly joinable under all possible total preorders, and we are done.

Together with this, some book-keeping for keeping track of completeness conditions is necessary. We know that for completeness to be guaranteed, the conditions in Definition 1 must hold. They automatically do if  $C$  is not a positive unit or if the rewrite happens on a strict subterm. We also know that after a term has been rewritten at least once, rewrites on that side are always complete (since it was rewritten to a smaller term). Therefore we store in the queue, together with the preorder, a flag in  $\mathcal{P}(\{\mathbf{L}, \mathbf{R}\})$  indicating on which sides does a top rewrite need to be checked for completeness. Initially the flag is  $\{\mathbf{L}\}$  if  $s \succ_t t$ ,  $\{\mathbf{R}\}$  if  $s \prec_t t$ ,  $\{\mathbf{L}, \mathbf{R}\}$  if  $s$  and  $t$  are incomparable, and  $\{\}$  if the clause is not a positive unit. When a rewrite at the top is attempted (say,  $l \approx r$  used to rewrite  $s = l\theta$  with  $t$  being the other side), if the flag for that side is set, then we check if  $l\theta \sqsubseteq l$  or  $r\theta \prec t$ . If this fails, the rewrite is rejected. Whenever a side is rewritten (at any position), the flag for that side is cleared.

The definition of `order_diff` is as follows. Let the transitive reduction of  $\succeq$  be represented by a set of links of the form  $x \succ y$  /  $x \sim y$ .

$$\text{order\_diff}(\succeq_1, \succeq_2) = \{\succeq^+ \mid \succeq \in \text{order\_diff}'(\succeq_1, \succeq_2^-)\}, \quad (11a)$$

$$\text{order\_diff}'(\succeq_1, \succeq_2^-) = \quad (11b)$$

$$\left\{ \begin{array}{l} \succeq_2^- = \{x \succ y\} \uplus \succeq_2'^- \Rightarrow \begin{cases} x \succ_1 y \Rightarrow \text{order\_diff}'(\succeq_1, \succeq_2'^-) \\ x \not\succ_1 y \Rightarrow \begin{cases} \{\succeq_1 \cup \{y \succ x\}, \succeq_1 \cup \{x \sim y\}\} \\ \cup \text{order\_diff}'(\succeq_1 \cup \{x \succ y\}, \succeq_2'^-) \end{cases} \end{cases} \\ \succeq_2^- = \{x \sim y\} \uplus \succeq_2'^- \Rightarrow \begin{cases} x \sim_1 y \Rightarrow \text{order\_diff}'(\succeq_1, \succeq_2'^-) \\ x \not\sim_1 y \Rightarrow \begin{cases} \{\succeq_1 \cup \{x \succ y\}, \succeq_1 \cup \{y \succ x\}\} \\ \cup \text{order\_diff}'(\succeq_1 \cup \{x \sim y\}, \succeq_2'^-) \end{cases} \end{cases} \\ \succeq_2^- = \emptyset \Rightarrow \emptyset. \end{array} \right.$$

where  $\succeq_1 \subseteq \succeq_2$ . In other words, we take a transitive reduction of  $\succeq_2$ , and for all links  $\ell$  in that reduction which are not part of  $\succeq_1$ , we return orders  $\succeq_1$  augmented with the reverse of  $\ell$  and recurse with  $\succeq_1 = \succeq_1 \cup \ell$ .

*Example 5.*

$\succeq_1$	$\succeq_2$	$\text{order\_diff}(\succeq_1, \succeq_2)$
$x \succ y$	$x \succ y \succ z \succ w$	$x \succ y \sim z, x \succ y \prec z, x \succ y \succ z \sim w, x \succ y \succ z \prec w$
$y \prec x \succ z$	$x \succ y \succ z$	$x \succ y \sim z, x \succ z \succ y$

**Theorem 3.** For all total  $\succeq_v^T \supseteq \succeq_1$ , there exists one and only one  $\succeq_i \in \{\succeq_2\} \cup \text{order\_diff}(\succeq_1, \succeq_2)$  such that  $\succeq_v^T \supseteq \succeq_i$ . For all  $\succeq_v^T \not\supseteq \succeq_1$ , there is no  $\succeq_i \in \{\succeq_2\} \cup \text{order\_diff}(\succeq_1, \succeq_2)$  such that  $\succeq_v^T \supseteq \succeq_i$ .

*Proof.* See full version of the paper [12].

An algorithm based on searching for rewrites in minimal extensions of a variable preorder (starting with minimal extensions of the bare term ordering,  $\succ_{t[\emptyset]}$ ), has several advantages. The main benefit of this approach is that, instead of imposing an a priori ordering on variables and then checking joinability under that ordering, we instead build a minimal ordering *while* searching for candidate unit equations to rewrite subterms of  $s, t$ . For instance, if two terms are *not* ground joinable, or not even rewritable in any  $\succ_{t[v]}$  where it was not rewritable in  $\succ_t$ , then an approach such as the one used in Avenhaus, Hillenbrand and Löchner [1] cannot detect this until it has extended the preorder arbitrarily to a total ordering, while our incremental algorithm immediately realises this. We should note that empirically this is what happens in most cases: most of the literals we check during a run are *not* ground joinable, so for practical performance it is essential to optimise this case.

**Theorem 4.** Algorithm 1 returns “Success” only if  $s \Downarrow t$  in  $C$  wrt.  $S$ .<sup>3</sup>

*Proof.* We will show that Algorithm 1 returns “Success” if and only if  $s \Downarrow_{\succ_{t[v^T]}} t$  for all total  $\succeq_v^T$  over  $\text{Vars}(s, t)$ , which implies  $s \Downarrow_{\succ_t} t$ .

When  $\langle \succeq_v, s, t, c \rangle$  is popped from  $V$ , we exhaustively reduce  $s, t$  via equations in  $S$  oriented wrt.  $\succ_{t[v]}$ , obtaining  $s^r, t^r$ . If  $s^r \sim_{t[v]} t^r$ , then  $s \Downarrow_{\succ_{t[v]}} t$ , and so  $s \Downarrow_{\succ_{t[v^T]}} t$  for all total  $\succeq_v^T \supseteq \succeq_v$ . If  $s^r \not\sim_{t[v]} t^r$ , we will attempt to rewrite one of  $s^r, t^r$  using *some* extended  $\succ_{t[v']}$  where  $\succeq'_v \supset \succeq_v$ . If this is impossible, then  $s \not\Downarrow_{\succ_{t[v']}} t$  for any  $\succeq'_v \supset \succeq_v$ , and therefore there exists at least one total  $\succeq_v^T$  such that  $s \not\Downarrow_{\succeq_v^T} t$ , and we return “Fail”.

If this is possible, then we repeat the process: we exhaustively reduce wrt.  $\succ_{t[v']}$ , obtaining  $s', t'$ . If  $s' \sim_{t[v']} t'$ , then we start again the process from the step where we attempt to rewrite via an extension of  $\succeq_v$ : we either find a rewrite with some  $\succ_{t[v'']}$  with  $\succeq''_v \supset \succeq'_v$ , and exhaustively normalise wrt.  $\succ_{t[v'']}$  obtaining  $s'', t''$ , etc., or we fail to do so and return “Fail”.

If in any such step (after exhaustively normalising wrt.  $\succ_{t[v']}$ ) we find  $s' \sim_{t[v']} t'$ , then  $s \Downarrow_{\succ_{t[v']}} t$ , and so  $s \Downarrow_{\succ_{t[v^T]}} t$  for all total  $\succeq_v^T \supseteq \succeq_v$ . Now at this point we must add back to the queue a set of preorders  $\succeq''_{v,i}$  such that: for all total  $\succeq_v^T \supseteq \succeq_v$ , either  $\succeq_v^T \supseteq \succeq'_v$  (proven to be  $\Downarrow$ ) or  $\succeq_v^T \supseteq$  some  $\succeq''_{v,i}$  (added to  $V$  to be checked). For efficiency, we would also like for there to be no overlap: no total  $\succeq_v^T \supseteq \succeq_v$  is an extension of more than one of  $\{\succeq_v, \succeq''_{v,1}, \dots\}$ .

This is true because of Theorem 3. So we add  $\{\langle \succeq''_{v,i}, s^r, t^r, c^r \rangle \mid \succeq''_{v,i} \in \text{order\_diff}(\succeq_v, \succeq'_v) \}$  to  $V$ , where  $c^r = c \setminus (\text{if } s^r \neq s \text{ then } \{L\} \text{ else } \{ \}) \setminus (\text{if } t^r \neq t \text{ then } \{R\} \text{ else } \{ \})$ . Note also that  $s \Downarrow_{\succ_{t[v]}} s^r$  and  $t \Downarrow_{\succ_{t[v]}} t^r$ , therefore also  $s \Downarrow_{\succ_{t[v'']}} s^r$  and  $t \Downarrow_{\succ_{t[v'']}} t^r$  if  $\succeq''_{v,i} \supset \succeq_v$ .

<sup>3</sup> Note that the other direction may not always hold, there are strongly ground joinable terms which are not detected by this method of analysing all preorders between variables, e.g.  $f(x, g(y)) \Downarrow f(g(y), x)$  wrt.  $S = \{f(x, y) \approx f(y, x)\}$ .

**Algorithm 1:** Incremental ground joinability test

---

**Input:** literal  $s \approx t \in C$ ; set of unorientable equations  $S$   
**Output:** whether  $s \Join t$  in  $C$  wrt.  $S$

**begin**

- $c \leftarrow \emptyset$  if  $C$  is not pos. unit,  $\{L\}$  if  $s \succ t$ ,  $\{R\}$  if  $s \prec t$ ,  $\{L, R\}$  otherwise
- $V \leftarrow \{(\emptyset, s, t, c)\}$
- while**  $V$  is not empty **do**
  - $\langle \succeq_v, s, t, c \rangle \leftarrow \text{pop from } V$
  - $s, t \leftarrow \text{normalise } s, t \text{ wrt. } \succ_{t[v]}, \text{ with completeness flag } c$
  - $c \leftarrow c \setminus (\{L\} \text{ if } s \text{ was changed}) \setminus (\{R\} \text{ if } t \text{ was changed})$
  - if**  $s \sim_{t[v]} t$  **then**
    - continue**
  - else**
    - $s', t', c' \leftarrow s, t, c$
    - while** there exists  $l \approx r \in S$  that can rewrite  $s'$  or  $t'$  wrt. some  $\succeq'_v \supset \succeq_v$ , with completeness flag  $c$  **do**
      - $s', t' \leftarrow \text{normalise } s', t' \text{ wrt. } \succ_{t[v]}, \text{ with completeness flag } c$
      - $c' \leftarrow c' \setminus (\{L\} \text{ if } s' \text{ was changed}) \setminus (\{R\} \text{ if } t' \text{ was changed})$
      - if**  $s' \sim_{t[v']} t'$  **then**
        - for**  $\succeq''_v$  in  $\text{order\_diff}(\succeq_v, \succeq'_v)$  **do** push  $\langle \succeq''_v, s, t, c \rangle$  to  $V$
        - break**
      - end**
      - $\succeq_v \leftarrow \succeq'_v$
    - else**
      - return** Fail
    - end**
  - end**
  - else**
    - return** Success
  - end**

**end**

**where** rewriting  $u$  in  $s, t$  wrt.  $\succ$  with completeness flag  $c$  succeeds **if**

  - (i)  $u$  is a strict subterm of  $s$  or  $t$ ,
  - (ii)  $u = s$  with  $L \notin c$ ,
  - (iii)  $u = t$  with  $R \notin c$ ,
  - (iv) instance  $l\sigma \approx r\sigma$  used to rewrite has  $l \sqsubset u$ ,
  - (v)  $u = s$  with  $r\sigma \prec t$ ,
  - (vi) or  $u = t$  with  $r\sigma \prec s$ .

**end**

---

During this whole process, any rewrites must pass a completeness test mentioned previously, such that the conditions in the definition of  $\Join$  hold. Let  $s_0, t_0$  be the original terms and  $s, t$  be the ones being rewritten and  $c$  the completeness flag. If the rewrite is at a strict subterm position, it succeeds by Definition 2. If the rewrite is at the top, then we check  $c$ . If  $L$  is unset ( $L \notin c$ ), then either  $s \succeq s_0 \prec t_0$  or  $s \prec s_0$  or the clause is not a positive unit, so we allow a rewrite at the top of  $s$ , again by Definition 2. If  $L$  is set ( $L \in c$ ), then an explicit check



must be done: we allow a rewrite at the top of  $s (= s_0)$  iff it is done by  $l\sigma \rightarrow r\sigma$  with  $l\sigma \sqsupset l$  or  $r\sigma \prec t_0$ . Respectively for R, with the roles of  $s$  and  $t$  swapped.

In short, we have shown that if  $\langle \succeq_v, s', t', c' \rangle$  is popped from  $V$ , then  $V$  is only ever empty, and so the algorithm only terminates with “Success”, if  $s' \not\prec_{\succ_{t[vT]}} t'$  for all total  $\succeq_v^T \supseteq \succeq_v$ . Since  $V$  is initialised with  $\langle \emptyset, s, t, c \rangle$ , then the algorithm only returns “Success” if  $s \not\prec_{\succ_{t[vT]}} t$  for all total  $\succeq_v^T$ .  $\square$

**Orienting via Extension of Variable Ordering.** In order to apply the ground joinability algorithm we need a way to check, for a given  $\succ_t$  and  $\succeq_v$  and some  $s, t$ , whether there exists a  $\succeq'_v \supset \succeq_v$  such that  $s \succ_{t[v']}\ t$ . Here we show how to do this when  $\succ_t$  is a Knuth-Bendix Ordering (KBO) [15].

Recall the definition of KBO. Let  $\succ_s$  be a partial order on symbols,  $w$  be an  $\mathbb{N}$ -valued weight function on symbols and variables, with the property that  $\exists m \forall x \in \mathcal{V}. w(x) = m, w(c) \geq m$  for all constants  $c$ , and there may only exist one unary symbol  $f$  with  $w(f) = 0$  and in this case  $f \succ_s g$  for all other symbols  $g$ . For terms, their weight is  $w(f(s_1, \dots)) = w(f) + w(s_1) + \dots$ . Let also  $|s|_x$  be the number of occurrences of  $x$  in  $s$ . Then

$$f(s_1, \dots) \succ_{\text{KBO}} g(t_1, \dots) \quad \text{iff} \quad \begin{cases} \text{either } w(f(s_1, \dots)) > w(g(t_1, \dots)), \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \text{and } f \succ_s g, \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \text{and } f = g, \\ \text{and } s_1, \dots \succ_{\text{KBO}^{\text{lex}}} t_1, \dots; \\ \text{and } \forall x \in \mathcal{V}. |f(\dots)|_x \geq |g(\dots)|_x. \end{cases} \quad (12a)$$

$$f(s_1, \dots) \succ_{\text{KBO}} x \quad \text{iff} \quad |f(s_1, \dots)|_x \geq 1. \quad (12b)$$

$$x \succ_{\text{KBO}} y \quad \text{iff} \quad \perp. \quad (12c)$$

The conditions on variable occurrences ensure that  $s \succ_{\text{KBO}} t \Rightarrow \forall \theta. s\theta \succ_{\text{KBO}} t\theta$ .

When we extend the order  $\succ_{\text{KBO}}$  with a variable preorder  $\succeq_v$ , the starting point is that  $x \succ_v y \Rightarrow x \succ_{\text{KBO}[v]} y$  and  $x \sim_v y \Rightarrow x \sim_{\text{KBO}[v]} y$ . Then, to ensure that all the properties of a simplification order (included the one mentioned above) hold, we arrive at the following definition (similar to [1]).

$$f(s_1, \dots) \succ_{\text{KBO}[v]} g(t_1, \dots) \quad \text{iff} \quad \begin{cases} \text{either } w(f(\dots)) > w(g(\dots)), \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \text{and } f \succ_s g, \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \text{and } f = g, \\ \text{and } s_1, \dots \succ_{\text{KBO}[v]^{\text{lex}}} t_1, \dots; \\ \text{and } \forall x \in \mathcal{V}. \sum_{y \succeq_v x} |f(\dots)|_y \\ \geq \sum_{y \succeq_v x} |g(\dots)|_y. \end{cases} \quad (13a)$$

$$f(s_1, \dots) \succ_{\text{KBO}[v]} x \quad \text{iff} \quad \exists y \succeq_v x. |f(s_1, \dots)|_y \geq 1. \quad (13b)$$

$$x \succ_{\text{KBO}[v]} y \quad \text{iff} \quad x \succ_v y. \quad (13c)$$

To check whether there exists a  $\succeq'_v \supset \succeq_v$  such that  $s \succ_{\text{KBO}[v']} t$ , we need to check whether there are some  $x \succ y$  or  $x = y$  relations that we can add to  $\succeq_v$  such that all the conditions above hold (and such that it still remains a valid preorder). Let us denote “there exists a  $\succeq'_v \supset \succeq_v$  such that  $s \succ_{\text{KBO}[v']} t$ ” by  $s \succ_{\text{KBO}[v,v']} t$ . Then the definition is

$$f(s_1, \dots) \succ_{\text{KBO}[v,v']} g(t_1, \dots) \quad \text{iff} \quad \left\{ \begin{array}{l} \text{either } w(f(\dots)) > w(g(\dots)), \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \quad \text{and } f \succ_s g, \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \quad \text{and } f = g, \\ \text{and } s_1, \dots \succ_{\text{KBOlex}} t_1, \dots; \\ \text{and } \exists x_1, y_1, \dots \\ \quad \succeq'_v = (\succeq_v \cup \{\langle x_1, y_1 \rangle, \dots\})^+ \text{ is a preorder} \\ \quad \text{such that } \forall x \in \mathcal{V}. \sum_{y \succeq'_v x} |f(\dots)|_y \\ \quad \geq \sum_{y \succeq'_v x} |g(\dots)|_y. \end{array} \right. \quad (14a)$$

$$f(s_1, \dots) \succ_{\text{KBO}[v,v']} x \quad \text{iff} \quad \left\{ \begin{array}{l} \exists y \not\succeq_v x. |f(s_1, \dots)|_y \geq 1, \\ \text{with } \succeq'_v = \succeq_v \cup \{x \succ y\} \\ \text{or } \succeq'_v = \succeq_v \cup \{x = y\}. \end{array} \right. \quad (14b)$$

$$x \succ_{\text{KBO}[v,v']} y \quad \text{iff} \quad \left\{ \begin{array}{l} x \not\succeq_v y \\ \text{with } \succeq'_v = \succeq_v \cup \{x \succ y\}. \end{array} \right. \quad (14c)$$

This check can be used in Algorithm 1 for finding extensions of variable orderings that orient rewrite rules allowing required normalisations.

### 4.3 Connectedness

Testing for joinability (i.e. demodulating to  $s \approx s$  or  $s \not\approx s$ ) and ground joinability (presented in the previous section) require that each step in proving them is done via an oriented instance of an equation in the set. However, we can weaken this restriction, if we also change the notion of redundancy being used.

As criteria for redundancy of a clause, finding either joinability or ground joinability of a literal in the clause means that the clause can be deleted or the literal removed from the clause (in case of a positive or negative literal, resp.) in any context, that is, we can for example add them to a set of deleted clauses, and for any new clause, if it appears in that set, then immediately remove it since we already saw that it is redundant. The criterion of connectedness [3, 21], however, is a criterion for redundancy of *inferences*. This means that a conclusion simplified by this criterion can be deleted (or rather, not added), but in that context only; if it ever comes up again as a conclusion of a different inference, then it is not necessarily also redundant. Connectedness was introduced in the context of equational completion, here we extend it to general clauses and show that it is a redundancy in the superposition calculus.

**Definition 4.** Terms  $s$  and  $t$  are *connected* under clauses  $U$  and unifier  $\rho$  wrt. a set of equations  $S$  if there exist terms  $v_1, \dots, v_n$ , equations  $l_1 \approx r_1, \dots, l_{n-1} \approx r_{n-1}$ , and substitutions  $\sigma_1, \dots, \sigma_{n-1}$  such that:

- (i)  $v_1 = s$  and  $v_n = t$ ,
- (ii) for all  $i \in 1, \dots, n-1$ , either  $v_{i+1} = v_i[l_i\sigma_i \mapsto r_i\sigma_i]$  or  $v_i = v_{i+1}[l_i\sigma_i \mapsto r_i\sigma_i]$ , with  $l_i \approx r_i \in S$ ,
- (iii) for all  $i \in 1, \dots, n-1$ , there exists  $w$  in  $\bigcup_{C \in U} \bigcup_{p \approx q \in C} \{p, q\}$ <sup>4</sup> such that for  $u_i \in \{l_i, r_i\}$ , either (a)  $u_i\sigma_i \prec w\rho$ , or (b)  $u_i\sigma_i = w\rho$  and either  $u_i \sqsubset w$  or  $w \in C$  such that  $C$  is not a positive unit.

**Theorem 5.** Superposition inferences of the form

$$\frac{l \approx r \vee C \quad s[u] \approx t \vee D}{(s[u \mapsto r] \approx t \vee C \vee D)\rho}, \quad \begin{array}{l} \text{where } \rho = \text{mgu}(l, u), \\ l\rho \not\approx r\rho, s\rho \not\approx t\rho, \\ \text{and } u \text{ not a variable,} \end{array} \quad (15)$$

where  $s[u \mapsto r]\rho$  and  $t\rho$  are connected under  $\{l \approx r \vee C, s \approx t \vee D\}$  and unifier  $\rho$  wrt. some set of clauses  $S$ , are redundant inferences wrt.  $S$ .

*Proof.* Let us denote  $s' = s[u \mapsto r]$ . Let also  $U = \{l \approx r \vee C, s \approx t \vee D\}$  and  $M = \bigcup_{C \in U} \bigcup_{p \approx q \in C} \{p, q\}$ . We will show that if  $s'\rho$  and  $t\rho$  are connected under  $U$  and  $\rho$ , by equations in  $S$ , then every instance of that inference obeys the condition for closure redundancy of an inference (see, Sect. 4), wrt.  $S$ .

Consider any  $(s' \approx t \vee C \vee D)\rho \cdot \theta$  where  $\theta \in \text{GSubs}(U\rho)$ . Either  $s'\rho\theta = t\rho\theta$ , and we are done (it follows from  $\emptyset$ ), or  $s'\rho\theta \succ t\rho\theta$ , or  $s'\rho\theta \prec t\rho\theta$ .

Consider the case  $s'\rho\theta \succ t\rho\theta$ . For all  $i \in 1, \dots, n-1$ , there exists a  $C' \in U$  and a  $w \in C'$  such that either (iii.a)  $l_i\sigma_i\theta \prec w\rho\theta$ , or (iii.b)  $l_i\sigma_i\theta = w\rho\theta$  and  $l_i \sqsubset v$ , or (iii.b)  $l_i\sigma_i\theta = w\rho\theta$  and  $C'$  is not a positive unit. Likewise for  $r_i$ . Therefore, for all  $i \in 1, \dots, n-1$ , there exists a  $C' \in U$  such that  $(l_i \approx r_i) \cdot \sigma_i\theta \prec C' \cdot \rho\theta$ . Since  $(t \approx t \vee \dots)\rho \cdot \theta$  is also smaller than  $(s' \approx t \vee \dots)\rho \cdot \theta$  and a tautology, then the instance  $(s' \approx t \vee \dots)\rho \cdot \theta$  of the conclusion follows from closures in  $\text{GClos}(S)$  such that each is smaller than one of  $(l \approx r \vee C) \cdot \rho\theta$ ,  $(s \approx t \vee D) \cdot \rho\theta$ .

In the case that  $s'\rho\theta \prec t\rho\theta$ , the same idea applies, but now it is  $(s' \approx s' \vee \dots)\rho \cdot \theta$  which is smaller than  $(s' \approx t \vee \dots)\rho \cdot \theta$  and is a tautology.

Therefore, we have shown that for all  $\theta \in \text{GSubs}((l \approx r \vee C)\rho, (s \approx t \vee D)\rho)$ , the instance  $(s' \approx t \vee \dots)\rho \cdot \theta$  of the conclusion follows from closures in  $\text{GClos}(S)$  which are all smaller than one of  $(l \approx r \vee C) \cdot \rho\theta$ ,  $(s \approx t \vee D) \cdot \rho\theta$ . Since any valid superposition inference with ground clauses has to have  $l = u$ , then any  $\theta' \in \text{GSubs}(l \approx r \vee C, s \approx t \vee D, (s' \approx t \vee C \vee D)\rho)$  such that the inference  $(l \approx r \vee C)\theta', (s \approx t \vee D)\theta' \vdash (s' \approx t \vee C \vee D)\rho\theta'$  is valid must have  $\theta' = \rho\theta''$ , since  $\rho$  is the most general unifier. Therefore, we have shown that for all  $\theta' \in \text{GSubs}(l \approx r \vee C, s \approx t \vee D, (s' \approx t \vee C \vee D)\rho)$  for which  $(l \approx r \vee C)\theta', (s \approx t \vee D)\theta' \vdash (s' \approx t \vee C \vee D)\rho\theta'$  is a valid superposition inference, the instance  $(s' \approx t \vee \dots)\rho \cdot \theta'$  of the conclusion follows from closures in  $\text{GClos}(S)$  which are all smaller than one of  $(l \approx r \vee C) \cdot \theta'$ ,  $(s \approx t \vee D) \cdot \theta'$ , so the inference is redundant.  $\square$

<sup>4</sup> That is, in the set of top-level terms of literals of clauses in  $U$ .

**Theorem 6.** Superposition inferences of the form

$$\frac{l \approx r \vee C \quad s[u] \not\approx t \vee D}{(s[u \mapsto r] \not\approx t \vee C \vee D)\rho}, \quad \begin{array}{l} \text{where } \rho = \text{mgu}(l, u), \\ l\rho \not\leq r\rho, s\rho \not\leq t\rho, \\ \text{and } u \text{ not a variable,} \end{array} \quad (16)$$

where  $s[u \mapsto r]\rho$  and  $t\rho$  are connected under  $\{l \approx r \vee C, s \not\approx t \vee D\}$  and unifier  $\rho$  wrt. some set of clauses  $S$ , are redundant inferences wrt.  $S \cup \{(C \vee D)\rho\}$ .

*Proof.* Analogously to the previous proof, we find that for all instances of the inference, the closure  $(s' \not\approx t \vee \dots)\rho \cdot \theta$  follows from smaller closure  $(t \not\approx t \vee \dots)\rho \cdot \theta$  or  $(s' \not\approx s' \vee \dots)\rho \cdot \theta$  and closures  $(l_i \approx r_i) \cdot \sigma_i \theta$  smaller than  $\max\{(l \approx r \vee C) \cdot \theta, (s \not\approx t \vee D) \cdot \theta, (s' \not\approx t \vee C \vee D)\rho \cdot \theta\}$ . But  $(t \not\approx t \vee C \vee D)\rho \cdot \theta$  and  $(s' \not\approx s' \vee C \vee D)\rho \cdot \theta$  both follow from smaller  $(C \vee D)\rho \cdot \theta$ , therefore the inference is redundant wrt.  $S \cup \{(C \vee D)\rho\}$ .  $\square$

#### 4.4 Ground Connectedness

Just as joinability can be generalised to ground joinability, so can connectedness be generalised to ground connectedness. Two terms  $s, t$  are *ground connected* under  $U$  and  $\rho$  wrt.  $S$  if, for all  $\theta \in \text{GSubs}(s, t)$ ,  $s\theta$  and  $t\theta$  are connected under  $D$  and  $\rho$  wrt.  $S$ . Analogously to strong ground joinability, we have that if  $s$  and  $t$  are connected using  $\succ_{t[v]}$  for all total  $\succeq_v$  over  $\text{Vars}(s, t)$ , then  $s$  and  $t$  are ground connected.

**Theorem 7.** Superposition inferences of the form

$$\frac{l \approx r \vee C \quad s[u] \approx t \vee D}{(s[u \mapsto r] \approx t \vee C \vee D)\rho}, \quad \begin{array}{l} \text{where } \rho = \text{mgu}(l, u), \\ l\rho \not\leq r\rho, s\rho \not\leq t\rho, \\ \text{and } u \text{ not a variable,} \end{array} \quad (17)$$

where  $s[u \mapsto r]\rho$  and  $t\rho$  are ground connected under  $\{l \approx r \vee C, s \approx t \vee D\}$  and unifier  $\rho$  wrt. some set of clauses  $S$ , are redundant inferences wrt.  $S$ .

**Theorem 8.** Superposition inferences of the form

$$\frac{l \approx r \vee C \quad s[u] \not\approx t \vee D}{(s[u \mapsto r] \not\approx t \vee C \vee D)\rho}, \quad \begin{array}{l} \text{where } \rho = \text{mgu}(l, u), \\ l\rho \not\leq r\rho, s\rho \not\leq t\rho, \\ \text{and } u \text{ not a variable,} \end{array} \quad (18)$$

where  $s[u \mapsto r]\rho$  and  $t\rho$  are ground connected under  $\{l \approx r \vee C, s \not\approx t \vee D\}$  and unifier  $\rho$  wrt. some set of clauses  $S$ , are redundant inferences wrt.  $S \cup \{(C \vee D)\rho\}$ .

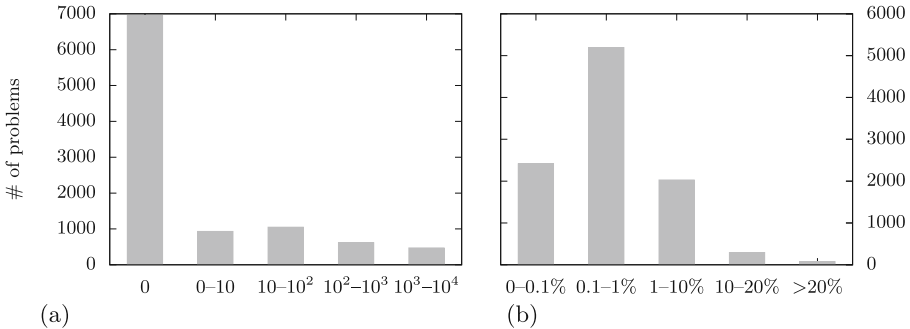
*Proof.* The proof of Theorem 7 and 8 is analogous to that of Theorem 5 and 6. The weakening of connectedness to ground connectedness only means that the proof of connectedness (e.g. the  $v_i, l_i \approx r_i, \sigma_i$ ) may be different for different ground instances. For all the steps in the proof to hold we only need that for all the instances  $\theta \in \text{GSubs}(l \approx r \vee C, s \approx t \vee D, (s[u \mapsto r] \approx t \vee C \vee D)\rho)$  of the inference,  $\theta = \sigma\theta'$  with  $\sigma \in \text{GSubs}(s[u \mapsto r]\rho, t\rho)$ , which is true.  $\square$

Discussion about the strategy for implementation of connectedness and ground connectedness is outside the scope of this paper.

## 5 Evaluation

We implemented ground joinability in a theorem prover for first-order logic, iProver [10,16].<sup>5</sup> iProver combines superposition, Inst-Gen, and resolution calculi. For superposition, iProver implements a range of simplifications including encompassment demodulation, AC normalisation [10], light normalisation [16], subsumption and subsumption resolution. We run our experiments over FOF problems of the TPTP v7.5 library [23] (17 348 problems) on a cluster of Linux servers with 3 GHz 11 core CPUs, 128 GB memory, with each problem running on a single core with a time limit of 300 s. We used a default strategy (which has not yet been fine-tuned after the introduction of ground joinability), with superposition enabled and the rest of the components disabled. With ground joinability enabled, iProver solved 133 problems more which it did not solve without ground joinability. Note that this excludes the contribution of AC ground joinability or encompassment demodulation [11] (always enabled).

Some of the problems are not interesting for this analysis because ground joinability is not even tried, either because they are solved before superposition saturation begins, or because they are ground. If we exclude these, we are left with 10 005 problems. Ground joinability is successfully used to eliminate clauses in 3057 of them (30.6%, Fig. 1a). This indicates that ground joinability is useful in many classes of problems, including in non-unit problems where it previously had never been used.



**Fig. 1.** (a) Clauses simplified by ground joinability. (b) % of runtime spent in gr. joinability

In terms of the performance impact of enabling ground joinability, we measure that among problems whose runtime exceeds 1s, only in 72 out of 8574 problems does the time spent inside the ground joinability algorithm exceed 20% of runtime, indicating that our incremental algorithm is efficient and suitable for broad application (Fig. 1b).

<sup>5</sup> iProver is available at <http://www.cs.man.ac.uk/~korovink/iprover>.

TPTP classifies problems by rating in  $[0,1]$ . Problems with rating  $\geq 0.9$  are considered to be very challenging. Problems with rating 1.0 have never been solved by any automated theorem prover. iProver using ground joinability solves 3 previously unsolved rating 1.0 problems, and 7 further problems with rating in  $[0.9,1.0[$  (Table 1). We note that some of these latter (e.g. LAT140-1, ROB018-10, REL045-1) were previously only solved by UEQ or SMT provers, but not by any full first-order prover.

**Table 1.** Hard or unsolved problems in TPTP, solved by iProver with ground joinability.

Name	Rating	Name	Rating
LAT140-1	0.90	ROB018-10	0.95
REL045-1	0.90	LCL477+1	0.97
LCL557+1	0.92	LCL478+1	1.00
LCL563+1	0.92	CSR039+6	1.00
LCL474+1	0.94	CSR040+6	1.00

## 6 Conclusion and Further Work

In this work we extended the superposition calculus with ground joinability and connectedness, and proved that these rules preserve completeness using a modified notion of redundancy, thus bringing for the first time these techniques for use in full first-order logic problems. We have also presented an algorithm for checking ground joinability which attempts to check as few variable preorders as possible.

Preliminary results show three things: (1) ground joinability is applicable in a sizeable number of problems across different domains, including in non-unit problems (where it was never applied before), (2) our proposed algorithm for checking ground joinability is efficient, with over  $\frac{3}{4}$  of problems spending less than 1% of runtime there, and (3) application of ground joinability in the superposition calculus of iProver improves overall performance, including discovering solutions to hitherto unsolved problems.

These results are promising, and further optimisations can be done. Immediate next steps include fine-tuning the implementation, namely adjusting the strategies and strategy combinations to make full use of ground joinability and connectedness. iProver uses a sophisticated heuristic system which has not yet been tuned for ground joinability and connectedness [14].

In terms of practical implementation of connectedness and ground connectedness, further research is needed on the interplay between those (criteria for redundancy of inferences) and joinability and ground joinability (criteria for redundancy of clauses).

On the theoretical level, recent work [24] provides a general framework for saturation theorem proving, and we will investigate how techniques developed in this paper can be incorporated into this framework.

## References

1. Avenhaus, J., Hillenbrand, T., Löchner, B.: On using ground joinable equations in equational theorem proving. *J. Symb. Comput.* **36**(1), 217–233 (2003). [https://doi.org/10.1016/S0747-7171\(03\)00024-5](https://doi.org/10.1016/S0747-7171(03)00024-5)
2. Baader, F., Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press (1998). ISBN 978-0521779203
3. Bachmair, L., Dershowitz, N.: Critical pair criteria for completion. *J. Symb. Comput.* **6**(1), 1–18 (1988). [https://doi.org/10.1016/S0747-7171\(88\)80018-X](https://doi.org/10.1016/S0747-7171(88)80018-X)
4. Bachmair, L., Dershowitz, N., Plaisted, D.A.: Completion without failure. In: Ait-Kaci, H., Nivat, M. (eds.) *Resolution of Equations in Algebraic Structures*, vol. II: *Rewriting Techniques*, pp. 1–30. Academic Press (1989). <https://doi.org/10.1016/B978-0-12-046371-8.50007-9>
5. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.* **4**(3), 217–247 (1994). <https://doi.org/10.1093/logcom/4.3.217>
6. Bachmair, L., Ganzinger, H., Lynch, C.A., Snyder, W.: Basic paramodulation. *Inf. Comput.* **121**(2), 172–192 (1995). <https://doi.org/10.1006/inco.1995.1131>. ISSN 0890-5401
7. Barthelemy, J.P.: An asymptotic equivalent for the number of total preorders on a finite set. *Discret. Math.* **29**(3), 311–313 (1980). [https://doi.org/10.1016/0012-365x\(80\)90159-4](https://doi.org/10.1016/0012-365x(80)90159-4)
8. Claessen, K., Smallbone, N.: Efficient encodings of first-order horn formulas in equational logic. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) *IJCAR 2018*. LNCS (LNAI), vol. 10900, pp. 388–404. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-94205-6\\_26](https://doi.org/10.1007/978-3-319-94205-6_26)
9. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. *Commun. ACM* **22**(8), 465–476 (1979). <https://doi.org/10.1145/359138.359142>
10. Duarte, A., Korovin, K.: Implementing superposition in iProver (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) *IJCAR 2020*. LNCS (LNAI), vol. 12167, pp. 388–397. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51054-1\\_24](https://doi.org/10.1007/978-3-030-51054-1_24)
11. Duarte, A., Korovin, K.: AC simplifications and closure redundancies in the superposition calculus. In: Das, A., Negri, S. (eds.) *TABLEAUX 2021*. LNCS (LNAI), vol. 12842, pp. 200–217. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-86059-2\\_12](https://doi.org/10.1007/978-3-030-86059-2_12)
12. Duarte, A., Korovin, K.: *Ground Joinability and Connectedness in the Superposition Calculus* (2022, to appear)
13. Hillenbrand, T., Buch, A., Vogt, R., Löchner, B.: Waldmeister—high-performance equational deduction. *J. Autom. Reason.* **18**(2), 265–270 (1997). <https://doi.org/10.1023/A:1005872405899>
14. Holden, E.K., Korovin, K.: Heterogeneous heuristic optimisation and scheduling for first-order theorem proving. In: Kamareddine, F., Sacerdoti Coen, C. (eds.) *CICM 2021*. LNCS (LNAI), vol. 12833, pp. 107–123. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-81097-9\\_8](https://doi.org/10.1007/978-3-030-81097-9_8)
15. Knuth, D.E., Bendix, P.: Simple word problems in universal algebras. In: Leech, J. (ed.) *Computational Problems in Abstract Algebra*, pp. 263–297. Pergamon (1970). <https://doi.org/10.1016/B978-0-08-012975-4.50028-X>
16. Korovin, K.: iProver—an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *IJCAR*

2008. LNCS (LNAI), vol. 5195, pp. 292–298. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-71070-7\\_24](https://doi.org/10.1007/978-3-540-71070-7_24)
17. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39799-8\\_1](https://doi.org/10.1007/978-3-642-39799-8_1)
  18. Martin, U., Nipkow, T.: Ordered rewriting and confluence. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 366–380. Springer, Heidelberg (1990). [https://doi.org/10.1007/3-540-52885-7\\_100](https://doi.org/10.1007/3-540-52885-7_100)
  19. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 2, pp. 371–443. Elsevier and MIT Press (2001). ISBN 0-444-50813-9
  20. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-45221-5\\_49](https://doi.org/10.1007/978-3-642-45221-5_49)
  21. Smallbone, N.: Twee: an equational theorem prover. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 602–613. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_35](https://doi.org/10.1007/978-3-030-79876-5_35)
  22. Sutcliffe, G.: The CADE ATP system competition–CASC. AI Mag. **37**(2), 99–101 (2016). <https://doi.org/10.1609/aimag.v37i2.2620>
  23. Sutcliffe, G.: The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017). <https://doi.org/10.1007/s10817-017-9407-7>
  24. Waldmann, U., Turret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 316–334. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51074-9\\_18](https://doi.org/10.1007/978-3-030-51074-9_18)
  25. Winkler, S., Moser, G.: MædMax: a maximal ordered completion tool. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 472–480. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-94205-6\\_31](https://doi.org/10.1007/978-3-319-94205-6_31)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.







# Connection-Minimal Abduction in $\mathcal{EL}$ via Translation to FOL

Fajar Haifani<sup>1,2(✉)</sup> , Patrick Koopmann<sup>3(✉)</sup> , Sophie Tourret<sup>1,4(✉)</sup> ,  
and Christoph Weidenbach<sup>1(✉)</sup>

<sup>1</sup> Max-Planck-Institut für Informatik, Saarland Informatics Campus,  
Saarbrücken, Germany

{f.haifani,c.weidenbach}@mpi-inf.mpg.de

<sup>2</sup> Graduate School of Computer Science, Saarbrücken, Germany

<sup>3</sup> TU Dresden, Dresden, Germany

patrick.koopmann@tu-dresden.de

<sup>4</sup> Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

sophie.tourret@inria.fr

**Abstract.** Abduction in description logics finds extensions of a knowledge base to make it entail an observation. As such, it can be used to explain why the observation does not follow, to repair incomplete knowledge bases, and to provide possible explanations for unexpected observations. We consider TBox abduction in the lightweight description logic  $\mathcal{EL}$ , where the observation is a concept inclusion and the background knowledge is a TBox, i.e., a set of concept inclusions. To avoid useless answers, such problems usually come with further restrictions on the solution space and/or minimality criteria that help sort the chaff from the grain. We argue that existing minimality notions are insufficient, and introduce connection minimality. This criterion follows Occam’s razor by rejecting hypotheses that use concept inclusions unrelated to the problem at hand. We show how to compute a special class of connection-minimal hypotheses in a sound and complete way. Our technique is based on a translation to first-order logic, and constructs hypotheses based on prime implicates. We evaluate a prototype implementation of our approach on ontologies from the medical domain.

## 1 Introduction

Ontologies are used in areas like biomedicine or the semantic web to represent and reason about terminological knowledge. They consist normally of a set of axioms formulated in a description logic (DL), giving definitions of concepts, or stating relations between them. In the lightweight description logic  $\mathcal{EL}$  [2], particularly used in the biomedical domain, we find ontologies that contain around a hundred thousand axioms. For instance, SNOMED CT<sup>1</sup> contains over 350,000 axioms, and the Gene Ontology GO<sup>2</sup> defines over 50,000 concepts. A central

<sup>1</sup> <https://www.snomed.org/>.

<sup>2</sup> <http://geneontology.org/>.

reasoning task for ontologies is to determine whether one concept is subsumed by another, a question that can be answered in polynomial time [1], and rather efficiently in practice using highly optimized description logic reasoners [29]. If the answer to this question is unexpected or hints at an error, a natural interest is in an explanation for that answer—especially if the ontology is complex. But whereas explaining entailments—i.e., explaining why a concept subsumption holds—is well-researched in the DL literature and integrated into standard ontology editors [21, 22], the problem of explaining non-entailments has received less attention, and there is no standard tool support. Classical approaches involve counter-examples [5], or *abduction*.

In abduction a non-entailment  $\mathcal{T} \not\models \alpha$ , for a TBox  $\mathcal{T}$  and an observation  $\alpha$ , is explained by providing a “missing piece”, the *hypothesis*, that, when added to the ontology, would entail  $\alpha$ . Thus it provides possible fixes in case the entailment should hold. In the DL context, depending on the shape of the observation, one distinguishes between concept abduction [6], ABox abduction [7–10, 12, 19, 24, 25, 30, 31], TBox abduction [11, 33] or knowledge base abduction [14, 26]. We are focusing here on TBox abduction, where the ontology and hypothesis are TBoxes and the observation is a concept inclusion (CI), i.e., a single TBox axiom.

To illustrate this problem, consider the following TBox, about academia,

$$\begin{aligned} \mathcal{T}_a = \{ & \exists \text{employment.ResearchPosition} \sqcap \exists \text{qualification.Diploma} \sqsubseteq \text{Researcher}, \\ & \exists \text{writes.ResearchPaper} \sqsubseteq \text{Researcher}, \text{Doctor} \sqsubseteq \exists \text{qualification.PhD}, \\ & \text{Professor} \equiv \text{Doctor} \sqcap \exists \text{employment.Chair}, \\ & \text{FundsProvider} \sqsubseteq \exists \text{writes.GrantApplication} \} \end{aligned}$$

that states, in natural language:

- “Being employed in a research position and having a qualifying diploma implies being a researcher.”
- “Writing a research paper implies being a researcher.”
- “Being a doctor implies holding a PhD qualification.”
- “Being a professor is being a doctor employed at a (university) chair.”
- “Being a funds provider implies writing grant applications.”

The observation  $\alpha_a = \text{Professor} \sqsubseteq \text{Researcher}$ , “Being a professor implies being a researcher”, does not follow from  $\mathcal{T}_a$  although it should. We can use TBox abduction to find different ways of recovering this entailment.

Commonly, to avoid trivial answers, the user provides syntactic restrictions on hypotheses, such as a set of abducible axioms to pick from [8, 30], a set of abducible predicates [25, 26], or patterns on the shape of the solution [11]. But even with those restrictions in place, there may be many possible solutions and, to find the ones with the best explanatory potential, syntactic criteria are usually combined with minimality criteria such as subset minimality, size minimality, or semantic minimality [7]. Even combined, these minimality criteria still retain a major flaw. They allow for explanations that go against the principle of parsimony, also known as Occam’s razor, in that they may contain concepts

that are completely unrelated to the problem at hands. As an illustration, let us return to our academia example. The TBoxes

$$\begin{aligned}\mathcal{H}_{a1} &= \{ \text{Chair} \sqsubseteq \text{ResearchPosition}, \text{PhD} \sqsubseteq \text{Diploma} \} \text{ and} \\ \mathcal{H}_{a2} &= \{ \text{Professor} \sqsubseteq \text{FundsProvider}, \text{GrantApplication} \sqsubseteq \text{ResearchPaper} \}\end{aligned}$$

are two hypotheses solving the TBox abduction problem involving  $\mathcal{T}_a$  and  $\alpha_a$ . Both of them are subset-minimal, have the same size, and are incomparable w.r.t. the entailment relation, so that traditional minimality criteria cannot distinguish them. However, intuitively, the second hypothesis feels more arbitrary than the first. Looking at  $\mathcal{H}_{a1}$ , Chair and ResearchPosition occur in  $\mathcal{T}_a$  in concept inclusions where the concepts in  $\alpha_a$  also occur, and both PhD and Diploma are similarly related to  $\alpha_a$  but via the role qualification. In contrast,  $\mathcal{H}_{a2}$  involves the concepts FundsProvider and GrantApplication that are not related to  $\alpha_a$  in any way in  $\mathcal{T}_a$ . In fact, any random concept inclusion  $A \sqsubseteq \exists \text{writes}.B$  in  $\mathcal{T}_a$  would lead to a hypothesis similar to  $\mathcal{H}_{a2}$  where  $A$  replaces FundsProvider and  $B$  replaces GrantApplication. Such explanations are not parsimonious.

We introduce a new minimality criterion called *connection minimality* that is parsimonious (Sect. 3), defined for the lightweight description logic  $\mathcal{EL}$ . This criterion characterizes hypotheses for  $\mathcal{T}$  and  $\alpha$  that connect the left- and right-hand sides of the observation  $\alpha$  without introducing spurious connections. To achieve this, every left-hand side of a CI in the hypothesis must follow from the left-hand side of  $\alpha$  in  $\mathcal{T}$ , and, taken together, all the right-hand sides of the CIs in the hypothesis must imply the right-hand side of  $\alpha$  in  $\mathcal{T}$ , as is the case for  $\mathcal{H}_{a1}$ . To compute connection-minimal hypotheses in practice, we present a technique based on first-order reasoning that proceeds in three steps (Sect. 4). First, we translate the abduction problem into a first-order formula  $\Phi$ . We then compute the prime implicates of  $\Phi$ , that is, a set of minimal logical consequences of  $\Phi$  that subsume all other consequences of  $\Phi$ . In the final step, we construct, based on those prime implicates, solutions to the original problem. We prove that all hypotheses generated in this way satisfy the connection minimality criterion, and that the method is complete for a relevant subclass of connection-minimal hypotheses. We use the SPASS theorem prover [34] as a restricted SOS-resolution [18, 35] engine for the computation of prime implicates in a prototype implementation (Sect. 5), and we present an experimental analysis of its performances on a set of bio-medical ontologies (Sect. 6). Our results indicate that our method can in many cases be applied in practice to compute connection-minimal hypotheses. A technical report companion of this paper includes all proofs as well as a detailed example of our method as appendices [16].

There are not many techniques that can handle TBox abduction in  $\mathcal{EL}$  or more expressive DLs [11, 26, 33]. In [11], instead of a set of abducibles, a set of *justification patterns* is given, in which the solutions have to fit. An arbitrary oracle function is used to decide whether a solution is admissible or not (which may use abducibles, justification patterns, or something else), and it is shown that deciding the existence of hypotheses is tractable. However, different to our approach, they only consider atomic CIs in hypotheses, while we also

allow for hypotheses involving conjunction. The setting from [33] also considers  $\mathcal{EL}$ , and abduction under various minimality notions such as subset minimality and size minimality. It presents practical algorithms, and an evaluation of an implementation for an always-true informativeness oracle (i.e., limited to subset minimality). Different to our approach, it uses an external DL reasoner to decide entailment relationships. In contrast, we present an approach that directly exploits first-order reasoning, and thus has the potential to be generalisable to more expressive DLs.

While dedicated resolution calculi have been used before to solve abduction in DLs [9, 26], to the best of our knowledge, the only work that relies on first-order reasoning for DL abduction is [24]. Similar to our approach, it uses SOS-resolution, but to perform ABox abduction for the more expressive DL  $\mathcal{ALC}$ . Apart from the different problem solved, in contrast to [24] we also provide a semantic characterization of the hypotheses generated by our method. We believe this characterization to be a major contribution of our paper. It provides an intuition of what parsimony is for this problem, independently of one's ease with first-order logic calculi, which should facilitate the adoption of this minimality criterion by the DL community. Thanks to this characterization, our technique is calculus agnostic. Any method to compute prime implicants in first-order logic can be a basis for our abduction technique, without additional theoretical work, which is not the case for [24]. Thus, abduction in  $\mathcal{EL}$  can benefit from the latest advances in prime implicants generation in first-order logic.

## 2 Preliminaries

We first recall the description logic  $\mathcal{EL}$  and its translation to first-order logic [2], as well as TBox abduction in this logic.

Let  $\mathbf{N}_C$  and  $\mathbf{N}_R$  be pair-wise disjoint, countably infinite sets of unary predicates called *atomic concepts* and of binary predicates called *roles*, respectively. Generally, we use letters  $A, B, E, F, \dots$  for atomic concepts, and  $r$  for roles, possibly annotated. Letters  $C, D$ , possibly annotated, denote  $\mathcal{EL}$  *concepts*, built according to the syntax rule

$$C ::= \top \mid A \mid C \sqcap C \mid \exists r.C .$$

We implicitly represent  $\mathcal{EL}$  conjunctions as sets, that is, without order, nested conjunctions, and multiple occurrences of a conjunct. We use  $\prod\{C_1, \dots, C_m\}$  to abbreviate  $C_1 \sqcap \dots \sqcap C_m$ , and identify the empty conjunction ( $m = 0$ ) with  $\top$ . An  $\mathcal{EL}$  *TBox*  $\mathcal{T}$  is a finite set of *concept inclusions* (CIs) of the form  $C \sqsubseteq D$ .

$\mathcal{EL}$  is a syntactic variant of a fragment of first-order logic that uses  $\mathbf{N}_C$  and  $\mathbf{N}_R$  as predicates. Specifically, TBoxes  $\mathcal{T}$  and CIs  $\alpha$  correspond to closed first-order formulas  $\pi(\mathcal{T})$  and  $\pi(\alpha)$  resp., while concepts  $C$  correspond to open formulas  $\pi(C, x)$  with a free variable  $x$ . In particular, we have

$$\begin{aligned}
\pi(\top, x) &:= \mathbf{true}, & \pi(\exists r.C, x) &:= \exists y.(r(x, y) \wedge \pi(C, y)), \\
\pi(A, x) &:= A(x), & \pi(C \sqsubseteq D) &:= \forall x.(\pi(C, x) \rightarrow \pi(D, x)), \\
\pi(C \sqcap D, x) &:= \pi(C, x) \wedge \pi(D, x), & \pi(\mathcal{T}) &:= \bigwedge \{\pi(\alpha) \mid \alpha \in \mathcal{T}\}.
\end{aligned}$$

As common, we often omit the  $\bigwedge$  in conjunctions  $\bigwedge \Phi$ , that is, we identify sets of formulas with the conjunction over those. The notions of a *term*  $t$ ; an *atom*  $P(\bar{t})$  where  $\bar{t}$  is a sequence of terms; a *positive literal*  $P(\bar{t})$ ; a *negative literal*  $\neg P(\bar{t})$ ; and a clause, Horn, definite, positive or negative, are defined as usual for first-order logic, and so are entailment and satisfaction of first-order formulas.

We identify CIs and TBoxes with their translation into first-order logic, and can thus speak of the entailment between formulas, CIs and TBoxes. When  $\mathcal{T} \models C \sqsubseteq D$  for some  $\mathcal{T}$ , we call  $C$  a *subsumee* of  $D$  and  $D$  a *subsumer* of  $C$ . We adhere here to the definition of the word “subsume”: “to include or contain something else”, although the terminology is reversed in first-order logic. We say two TBoxes  $\mathcal{T}_1, \mathcal{T}_2$  are *equivalent*, denoted  $\mathcal{T}_1 \equiv \mathcal{T}_2$  iff  $\mathcal{T}_1 \models \mathcal{T}_2$  and  $\mathcal{T}_2 \models \mathcal{T}_1$ . For example  $\{D \sqsubseteq C_1, \dots, D \sqsubseteq C_n\} \equiv \{D \sqsubseteq C_1 \sqcap \dots \sqcap C_n\}$ . It is well known that, due to the absence of concept negation, every  $\mathcal{EL}$  TBox is consistent.

The abduction problem we are concerned with in this paper is the following:

**Definition 1.** *An  $\mathcal{EL}$  TBox abduction problem (shortened to abduction problem) is a tuple  $\langle \mathcal{T}, \Sigma, C_1 \sqsubseteq C_2 \rangle$ , where  $\mathcal{T}$  is a TBox called the background knowledge,  $\Sigma$  is a set of atomic concepts called the abducible signature, and  $C_1 \sqsubseteq C_2$  is a CI called the observation, s.t.  $\mathcal{T} \not\models C_1 \sqsubseteq C_2$ . A solution to this problem is a TBox*

$$\mathcal{H} \subseteq \{A_1 \sqcap \dots \sqcap A_n \sqsubseteq B_1 \sqcap \dots \sqcap B_m \mid \{A_1, \dots, A_n, B_1, \dots, B_m\} \subseteq \Sigma\}$$

where  $m > 0, n \geq 0$  and such that  $\mathcal{T} \cup \mathcal{H} \models C_1 \sqsubseteq C_2$  and, for all CIs  $\alpha \in \mathcal{H}$ ,  $\mathcal{T} \not\models \alpha$ . A solution to an abduction problem is called a *hypothesis*.

For example,  $\mathcal{H}_{a1}$  and  $\mathcal{H}_{a2}$  are solutions for  $\langle \mathcal{T}_a, \Sigma, \alpha_a \rangle$ , as long as  $\Sigma$  contains all the atomic concepts that occur in them. Note that in our setting, as in [6, 33], concept inclusions in a hypothesis are *flat*, i.e., they contain no existential role restrictions. While this restricts the solution space for a given problem, it is possible to bypass this limitation in a targeted way, by introducing fresh atomic concepts equivalent to a concept of interest. We exclude the consistency requirement  $\mathcal{T} \cup \mathcal{H} \not\models \perp$ , that is given in other definitions of DL abduction problem [25], since  $\mathcal{EL}$  TBoxes are always consistent. We also allow  $m > 1$  instead of the usual  $m = 1$ . This produces the same hypotheses modulo equivalence.

For simplicity, we assume in the following that the concepts  $C_1$  and  $C_2$  in the abduction problem are atomic. We can always introduce fresh atomic concepts  $A_1$  and  $A_2$  with  $A_1 \sqsubseteq C_1$  and  $C_2 \sqsubseteq A_2$  to solve the problem for complex concepts.

Common minimality criteria include *subset* minimality, *size* minimality and *semantic* minimality, that respectively favor  $\mathcal{H}$  over  $\mathcal{H}'$  if:  $\mathcal{H} \subsetneq \mathcal{H}'$ ; the number of atomic concepts in  $\mathcal{H}$  is smaller than in  $\mathcal{H}'$ ; and if  $\mathcal{H} \models \mathcal{H}'$  but  $\mathcal{H}' \not\models \mathcal{H}$ .

### 3 Connection-Minimal Abduction

To address the lack of parsimony of common minimality criteria, illustrated in the academia example, we introduce *connection* minimality. Intuitively, connection minimality only accepts those hypotheses that ensure that every CI in the hypothesis is connected to both  $C_1$  and  $C_2$  in  $\mathcal{T}$ , as is the case for  $\mathcal{H}_{a1}$  in the academia example. The definition of connection minimality is based on the following ideas: 1) Hypotheses for the abduction problem should create a *connection* between  $C_1$  and  $C_2$ , which can be seen as a concept  $D$  that satisfies  $\mathcal{T} \cup \mathcal{H} \models C_1 \sqsubseteq D$ ,  $D \sqsubseteq C_2$ . 2) To ensure parsimony, we want this connection to be based on concepts  $D_1$  and  $D_2$  for which we already have  $\mathcal{T} \models C_1 \sqsubseteq D_1$ ,  $D_2 \sqsubseteq C_2$ . This prevents the introduction of unrelated concepts in the hypothesis. Note however that  $D_1$  and  $D_2$  can be complex, thus the connection from  $C_1$  to  $D_1$  (resp.  $D_2$  to  $C_2$ ) can be established by arbitrarily long chains of concept inclusions. 3) We additionally want to make sure that the connecting concepts are not more complex than necessary, and that  $\mathcal{H}$  only contains CIs that directly connect parts of  $D_2$  to parts of  $D_1$  by closely following their structure.

To address point 1), we simply introduce connecting concepts formally.

**Definition 2.** *Let  $C_1$  and  $C_2$  be concepts. A concept  $D$  connects  $C_1$  to  $C_2$  in  $\mathcal{T}$  if and only if  $\mathcal{T} \models C_1 \sqsubseteq D$  and  $\mathcal{T} \models D \sqsubseteq C_2$ .*

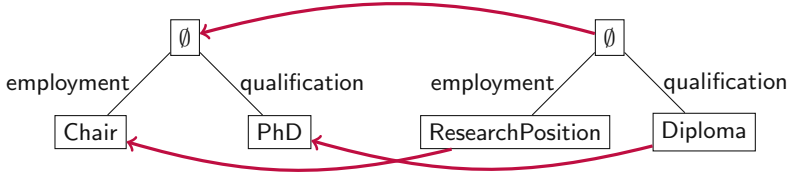
Note that if  $\mathcal{T} \models C_1 \sqsubseteq C_2$  then both  $C_1$  and  $C_2$  are connecting concepts from  $C_1$  to  $C_2$ , and if  $\mathcal{T} \not\models C_1 \sqsubseteq C_2$ , the case of interest, neither of them are.

To address point 2), we must capture *how* a hypothesis creates the connection between the concepts  $C_1$  and  $C_2$ . As argued above, this is established via concepts  $D_1$  and  $D_2$  that satisfy  $\mathcal{T} \models C_1 \sqsubseteq D_1$ ,  $D_2 \sqsubseteq C_2$ . Note that having only two concepts  $D_1$  and  $D_2$  is exactly what makes the approach parsimonious. If there was only one concept,  $C_1$  and  $C_2$  would already be connected, and as soon as there are more than two concepts, hypotheses start becoming more arbitrary: for a very simple example with unrelated concepts, assume given a TBox that entails  $\text{Lion} \sqsubseteq \text{Felidae}$ ,  $\text{Mammal} \sqsubseteq \text{Animal}$  and  $\text{House} \sqsubseteq \text{Building}$ . A possible hypothesis to explain  $\text{Lion} \sqsubseteq \text{Animal}$  is  $\{\text{Felidae} \sqsubseteq \text{House}, \text{Building} \sqsubseteq \text{Mammal}\}$  but this explanation is more arbitrary than  $\{\text{Felidae} \sqsubseteq \text{Mammal}\}$ —as is the case when comparing  $\mathcal{H}_{a2}$  with  $\mathcal{H}_{a1}$  in the academia example—because of the lack of connection of  $\text{House} \sqsubseteq \text{Building}$  with both  $\text{Lion}$  and  $\text{Animal}$ . Clearly this CI could be replaced by any other CI entailed by  $\mathcal{T}$ , which is what we want to avoid.

We can represent the structure of  $D_1$  and  $D_2$  in graphs by using  $\mathcal{EL}$  *description trees*, originally from Baader et al. [3].

**Definition 3.** *An  $\mathcal{EL}$  description tree is a finite labeled tree  $\mathfrak{T} = (V, E, v_0, l)$  where  $V$  is a set of nodes with root  $v_0 \in V$ , the nodes  $v \in V$  are labeled with  $l(v) \subseteq \mathbf{N}_C$ , and the (directed) edges  $vrw \in E$  are such that  $v, w \in V$  and are labeled with  $r \in \mathbf{N}_R$ .*

Given a tree  $\mathfrak{T} = (V, E, v_0, l)$  and  $v \in V$ , we denote by  $\mathfrak{T}(v)$  the subtree of  $\mathfrak{T}$  that is rooted in  $v$ . If  $l(v_0) = \{A_1, \dots, A_k\}$  and  $v_1, \dots, v_n$  are all the children of  $v_0$ , we



**Fig. 1.** Description trees of  $D_1$  (left) and  $D_2$  (right).

can define the concept represented by  $\mathfrak{T}$  recursively using  $C_{\mathfrak{T}} = A_1 \sqcap \dots \sqcap A_k \sqcap \exists r_1.C_{\mathfrak{T}(v_1)} \sqcap \dots \sqcap \exists r_l.C_{\mathfrak{T}(v_l)}$  where for  $j \in \{1, \dots, n\}$ ,  $v_0 r_j v_j \in E$ . Conversely, we can define  $\mathfrak{T}_C$  for a concept  $C = A_1 \sqcap \dots \sqcap A_k \sqcap \exists r_1.C_1 \sqcap \dots \sqcap \exists r_n.C_n$  inductively based on the pairwise disjoint description trees  $\mathfrak{T}_{C_i} = \{V_i, E_i, v_i, l_i\}$ ,  $i \in \{1, \dots, n\}$ . Specifically,  $\mathfrak{T}_C = (V_C, E_C, v_C, l_C)$ , where

$$\begin{aligned} V_C &= \{v_0\} \cup \bigcup_{i=1}^n V_i, & l_C(v) &= l_i(v) \text{ for } v \in V_i, \\ E_C &= \{v_0 r_i v_i \mid 1 \leq i \leq n\} \cup \bigcup_{i=1}^n E_i, & l_C(v_0) &= \{A_1, \dots, A_k\}. \end{aligned}$$

If  $\mathcal{T} = \emptyset$ , then subsumption between  $\mathcal{EL}$  concepts is characterized by the existence of a homomorphism between the corresponding description trees [3]. We generalise this notion to also take the TBox into account.

**Definition 4.** Let  $\mathfrak{T}_1 = (V_1, E_1, v_0, l_1)$  and  $\mathfrak{T}_2 = (V_2, E_2, w_0, l_2)$  be two description trees and  $\mathcal{T}$  a TBox. A mapping  $\phi : V_2 \rightarrow V_1$  is a  $\mathcal{T}$ -homomorphism from  $\mathfrak{T}_2$  to  $\mathfrak{T}_1$  if and only if the following conditions are satisfied:

1.  $\phi(w_0) = v_0$
2.  $\phi(v)r\phi(w) \in E_1$  for all  $vrw \in E_2$
3. for every  $v \in V_1$  and  $w \in V_2$  with  $v = \phi(w)$ ,  $\mathcal{T} \models \bigwedge l_1(v) \sqsubseteq \bigwedge l_2(w)$

If only 1 and 2 are satisfied, then  $\phi$  is called a weak homomorphism.

$\mathcal{T}$ -homomorphisms for a given TBox  $\mathcal{T}$  capture subsumption w.r.t.  $\mathcal{T}$ . If there exists a  $\mathcal{T}$ -homomorphism  $\phi$  from  $\mathfrak{T}_2$  to  $\mathfrak{T}_1$ , then  $\mathcal{T} \models C_{\mathfrak{T}_1} \sqsubseteq C_{\mathfrak{T}_2}$ . This can be shown easily by structural induction using the definitions [16]. The weak homomorphism is the structure on which a  $\mathcal{T}$ -homomorphism can be built by adding some hypothesis  $\mathcal{H}$  to  $\mathcal{T}$ . It is used to reveal missing links between a subsumee  $D_2$  of  $C_2$  and a subsumer  $D_1$  of  $C_1$ , that can be added using  $\mathcal{H}$ .

*Example 5.* Consider the concepts

$$\begin{aligned} D_1 &= \exists \text{employment.Chair} \sqcap \exists \text{qualification.Phd} \\ D_2 &= \exists \text{employment.ResearchPosition} \sqcap \exists \text{qualification.Diploma} \end{aligned}$$

from the academia example. Figure 1 illustrates description trees for  $D_1$  (left) and  $D_2$  (right). The curved arrows show a weak homomorphism from  $\mathfrak{T}_{D_2}$  to  $\mathfrak{T}_{D_1}$  that can be strengthened into a  $\mathcal{T}$ -homomorphism for some TBox  $\mathcal{T}$  that corresponds to the set of CIs in  $\mathcal{H}_{a1} \cup \{\top \sqsubseteq \top\}$ . The figure can also be used to

illustrate what we mean by connection minimality: in order to create a connection between  $D_1$  and  $D_2$ , we should *only* add the CIs from  $\mathcal{H}_{a1} \cup \{\top \sqsubseteq \top\}$  *unless* they are already entailed by  $\mathcal{T}_a$ . In practice, this means the weak homomorphism from  $D_2$  to  $D_1$  becomes a  $(\mathcal{T}_a \cup \mathcal{H}_{a1})$ -homomorphism.

To address point 3), we define a partial order  $\preceq_\square$  on concepts, s.t.  $C \preceq_\square D$  if we can turn  $D$  into  $C$  by removing conjuncts in subexpressions, e.g.,  $\exists r'.B \preceq_\square \exists r.A \sqcap \exists r'.(B \sqcap B')$ . Formally, this is achieved by the following definition.

**Definition 6.** *Let  $C$  and  $D$  be arbitrary concepts. Then  $C \preceq_\square D$  if either:*

- $C = D$ ,
- $D = D' \sqcap D''$ , and  $C \preceq_\square D'$ , or
- $C = \exists r.C'$ ,  $D = \exists r.D'$  and  $C' \preceq_\square D'$ .

We can finally capture our ideas on connection minimality formally.

**Definition 7 (Connection-Minimal Abduction).** *Given an abduction problem  $\langle \mathcal{T}, \Sigma, C_1 \sqsubseteq C_2 \rangle$ , a hypothesis  $\mathcal{H}$  is connection-minimal if there exist concepts  $D_1$  and  $D_2$  built over  $\Sigma \cup \mathbf{N}_R$  and a mapping  $\phi$  satisfying each of the following conditions:*

1.  $\mathcal{T} \models C_1 \sqsubseteq D_1$ ,
2.  $D_2$  is a  $\preceq_\square$ -minimal concept s.t.  $\mathcal{T} \models D_2 \sqsubseteq C_2$ ,
3.  $\phi$  is a weak homomorphism from the tree  $\mathfrak{T}_{D_2} = (V_2, E_2, w_0, l_2)$  to the tree  $\mathfrak{T}_{D_1} = (V_1, E_1, v_0, l_1)$ , and
4.  $\mathcal{H} = \{\sqcap l_1(\phi(w)) \sqsubseteq \sqcap l_2(w) \mid w \in V_2 \wedge \mathcal{T} \not\models \sqcap l_1(\phi(w)) \sqsubseteq \sqcap l_2(w)\}$ .

$\mathcal{H}$  is additionally called *packed* if the left-hand sides of the CIs in  $\mathcal{H}$  cannot hold more conjuncts than they do, which is formally stated as: for  $\mathcal{H}$ , there is no  $\mathcal{H}'$  defined from the same  $D_2$  and a  $D'_1$  and  $\phi'$  s.t. there is a node  $w \in V_2$  for which  $l_1(\phi(w)) \subsetneq l'_1(\phi'(w))$  and  $l_1(\phi(w)) = l'_1(\phi'(w'))$  for  $w' \neq w$ .

Straightforward consequences of Definition 7 include that  $\phi$  is a  $(\mathcal{T} \cup \mathcal{H})$ -homomorphism from  $\mathfrak{T}_{D_2}$  to  $\mathfrak{T}_{D_1}$  and that  $D_1$  and  $D_2$  are connecting concepts from  $C_1$  to  $C_2$  in  $\mathcal{T} \cup \mathcal{H}$  so that  $\mathcal{T} \cup \mathcal{H} \models C_1 \sqsubseteq C_2$  as wanted [16]. With the help of Fig. 1 and Example 5, one easily establishes that hypothesis  $\mathcal{H}_{a1}$  is connection-minimal—and even packed. Connection-minimality rejects  $\mathcal{H}_{a2}$ , as a single  $\mathcal{T}'$ -homomorphism for some  $\mathcal{T}'$  between two concepts  $D_1$  and  $D_2$  would be insufficient: we would need two weak homomorphisms, one linking Professor to FundsProvider and another linking  $\exists \text{writes.GrantApplication}$  to  $\exists \text{writes.ResearchPaper}$ .

## 4 Computing Connection-Minimal Hypotheses Using Prime Implicates

To compute connection-minimal hypotheses in practice, we propose a method based on first-order prime implicates, that can be derived by resolution. We



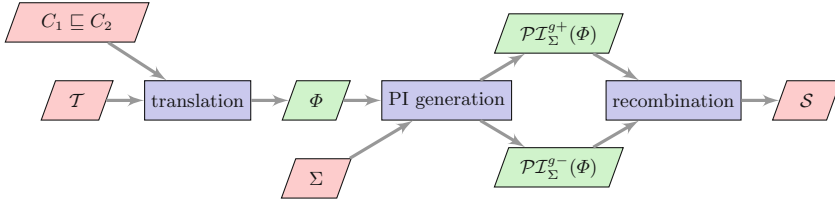


Fig. 2.  $\mathcal{EL}$  abduction using prime implicate generation in FOL.

assume the reader is familiar with the basics of first-order resolution, and do not reintroduce notions of clauses, Skolemization and resolution inferences here (for details, see [4]). In our context, every term is built on variables, denoted  $x, y$ , a single constant  $\mathbf{sk}_0$  and unary Skolem functions usually denoted  $\mathbf{sk}$ , possibly annotated. Prime implicates are defined as follows.

**Definition 8 (Prime Implicate).** *Let  $\Phi$  be a set of clauses. A clause  $\varphi$  is an implicate of  $\Phi$  if  $\Phi \models \varphi$ . Moreover  $\varphi$  is prime if for any other implicate  $\varphi'$  of  $\Phi$  s.t.  $\varphi' \models \varphi$ , it also holds that  $\varphi \models \varphi'$ .*

Let  $\Sigma \subseteq \mathbf{N}_C$  be a set of unary predicates. Then  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$  denotes the set of all positive ground prime implicates of  $\Phi$  that only use predicate symbols from  $\Sigma \cup \mathbf{N}_R$ , while  $\mathcal{PT}_{\Sigma}^{g-}(\Phi)$  denotes the set of all negative ground prime implicates of  $\Phi$  that only use predicates symbols from  $\Sigma \cup \mathbf{N}_R$ .

*Example 9.* Given a set of clauses  $\Phi = \{A_1(\mathbf{sk}_0), \neg B_1(\mathbf{sk}_0), \neg A_1(x) \vee r(x, \mathbf{sk}(x)), \neg A_1(x) \vee A_2(\mathbf{sk}(x)), \neg B_2(x) \vee \neg r(x, y) \vee \neg B_3(y) \vee B_1(x)\}$ , the ground prime implicates of  $\Phi$  for  $\Sigma = \mathbf{N}_C$  are, on the positive side,  $\mathcal{PT}_{\Sigma}^{g+}(\Phi) = \{A_1(\mathbf{sk}_0), A_2(\mathbf{sk}(\mathbf{sk}_0)), r(\mathbf{sk}_0, \mathbf{sk}(\mathbf{sk}_0))\}$  and, on the negative side,  $\mathcal{PT}_{\Sigma}^{g-}(\Phi) = \{\neg B_1(\mathbf{sk}_0), \neg B_2(\mathbf{sk}_0) \vee \neg B_3(\mathbf{sk}(\mathbf{sk}_0))\}$ . They are implicates because all of them are entailed by  $\Phi$ . For a ground implicate  $\varphi$ , another ground implicate  $\varphi'$  such that  $\varphi' \models \varphi$  and  $\varphi \not\models \varphi'$  can only be obtained from  $\varphi$  by dropping literals. Such an operation does not produce another implicate for any of the clauses presented above as belonging to  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$  and  $\mathcal{PT}_{\Sigma}^{g-}(\Phi)$ , thus they really are all prime.

To generate hypotheses, we translate the abduction problem into a set of first-order clauses, from which we can infer prime implicates that we then combine to obtain the result as illustrated in Fig. 2. In more details: We first translate the problem into a set  $\Phi$  of Horn clauses. Prime implicates can be computed using an off-the-shelf tool [13, 28] or, in our case, a slight extension of the resolution-based version of the SPASS theorem prover [34] using the set-of-support strategy and some added features described in Sect. 5. Since  $\Phi$  is Horn,  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$  contains only unit clauses. A final recombination step looks at the clauses in  $\mathcal{PT}_{\Sigma}^{g-}(\Phi)$  one after the other. These correspond to candidates for the connecting concepts  $D_2$  of Definition 7. Recombination attempts to match each literal in one such clause with unit clauses from  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$ . If such a match is possible, it produces a

suitable  $D_1$  to match  $D_2$ , and allows the creation of a solution to the abduction problem. The set  $\mathcal{S}$  contains all the hypotheses thus obtained.

In what follows, we present our translation of abduction problems into first-order logic and formalize the construction of hypotheses from the prime implicates of this translation. We then show how to obtain termination for the prime implicate generation process with soundness and completeness guarantees on the solutions computed.

**Abduction Method.** We assume the  $\mathcal{EL}$  TBox in the input is in normal form as defined, e.g., by Baader et al. [2]. Thus every CI is of one of the following forms:

$$A \sqsubseteq B \quad A_1 \sqcap A_2 \sqsubseteq B \quad \exists r.A \sqsubseteq B \quad A \sqsubseteq \exists r.B$$

where  $A, A_1, A_2, B \in \mathbf{N}_C \cup \{\top\}$ .

The use of normalization is justified by the following lemma.

**Lemma 10.** *For every  $\mathcal{EL}$  TBox  $\mathcal{T}$ , we can compute in polynomial time an  $\mathcal{EL}$  TBox  $\mathcal{T}'$  in normal form such that for every other TBox  $\mathcal{H}$  and every CI  $C \sqsubseteq D$  that use only names occurring in  $\mathcal{T}$ , we have  $\mathcal{T} \cup \mathcal{H} \models C \sqsubseteq D$  iff  $\mathcal{T}' \cup \mathcal{H} \models C \sqsubseteq D$ .*

After the normalisation, we eliminate occurrences of  $\top$ , replacing this concept everywhere by the fresh atomic concept  $A_\top$ . We furthermore add  $\exists r.A_\top \sqsubseteq A_\top$  and  $B \sqsubseteq A_\top$  in  $\mathcal{T}$  for every role  $r$  and atomic concept  $B$  occurring in  $\mathcal{T}$ . This simulates the semantics of  $\top$  for  $A_\top$ , namely the implicit property that  $C \sqsubseteq \top$  holds for any  $C$  no matter what the TBox is. In particular, this ensures that whenever there is a positive prime implicate  $B(t)$  or  $r(t, t')$ ,  $A_\top(t)$  also becomes a prime implicate. Note that normalisation and  $\top$  elimination extend the signature, and thus potentially the solution space of the abduction problem. This is remedied by intersecting the set of abducible predicates  $\Sigma$  with the signature of the original input ontology. We assume that  $\mathcal{T}$  is in normal form and without  $\top$  in the rest of the paper.

We denote by  $\mathcal{T}^-$  the result of renaming all atomic concepts  $A$  in  $\mathcal{T}$  using fresh *duplicate* symbols  $A^-$ . This renaming is done only on concepts but not on roles, and on  $C_2$  but not on  $C_1$  in the observation. This ensures that the literals in a clause of  $\mathcal{PT}_\Sigma^{g-}(\Phi)$  all relate to the conjuncts of a  $\preceq_\sqcap$ -minimal subsumee of  $C_2$ . Without it, some of these conjuncts would not appear in the negative implicates due to the presence of their positive counterparts as atoms in  $\mathcal{PT}_\Sigma^{g+}(\Phi)$ . The translation of the abduction problem  $\langle \mathcal{T}, \Sigma, C_1 \sqsubseteq C_2 \rangle$  is defined as the Skolemization of

$$\pi(\mathcal{T} \uplus \mathcal{T}^-) \wedge \neg\pi(C_1 \sqsubseteq C_2^-)$$

where  $\mathbf{sk}_0$  is used as the unique fresh Skolem constant such that the Skolemization of  $\neg\pi(C_1 \sqsubseteq C_2^-)$  results in  $\{C_1(\mathbf{sk}_0), \neg C_2^-(\mathbf{sk}_0)\}$ . This translation is usually denoted  $\Phi$  and always considered in clausal normal form.

**Theorem 11.** *Let  $\langle \mathcal{T}, \Sigma, C_1 \sqsubseteq C_2 \rangle$  be an abduction problem and  $\Phi$  be its first-order translation. Then, a TBox  $\mathcal{H}'$  is a packed connection-minimal solution to the problem if and only if an equivalent hypothesis  $\mathcal{H}$  can be constructed from non-empty sets  $\mathcal{A}$  and  $\mathcal{B}$  of atoms verifying:*

- $\mathcal{B} = \{B_1(t_1), \dots, B_m(t_m)\}$  s.t.  $(\neg B_1^-(t_1) \vee \dots \vee \neg B_m^-(t_m)) \in \mathcal{PT}_{\Sigma}^{g-}(\Phi)$ ,
- for all  $t \in \{t_1, \dots, t_m\}$  there exists an  $A$  s.t.  $A(t) \in \mathcal{PT}_{\Sigma}^{g+}(\Phi)$ ,
- $\mathcal{A} = \{A(t) \in \mathcal{PT}_{\Sigma}^{g+}(\Phi) \mid t \text{ is one of } t_1, \dots, t_m\}$ , and
- $\mathcal{H} = \{C_{\mathcal{A},t} \sqsubseteq C_{\mathcal{B},t} \mid t \text{ is one of } t_1, \dots, t_m \text{ and } C_{\mathcal{B},t} \not\sqsubseteq_{\sqcap} C_{\mathcal{A},t}\}$ , where  $C_{\mathcal{A},t} = \bigcap_{A(t) \in \mathcal{A}} A$  and  $C_{\mathcal{B},t} = \bigcap_{B(t) \in \mathcal{B}} B$ .

We call the hypotheses that are constructed as in Theorem 11 *constructible*. This theorem states that every packed connection-minimal hypothesis is equivalent to a constructible hypothesis and vice versa. A constructible hypothesis is built from the concepts in *one* negative prime implicate in  $\mathcal{PT}_{\Sigma}^{g-}(\Phi)$  and *all* matching concepts from prime implicates in  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$ . The matching itself is determined by the Skolem terms that occur in all these clauses. The subterm relation between the terms of the clauses in  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$  and  $\mathcal{PT}_{\Sigma}^{g-}(\Phi)$  is the same as the ancestor relation in the description trees of subsumers of  $C_1$  and subsumees of  $C_2$  respectively. The terms matching in positive and negative prime implicates allow us to identify where the missing entailments between a subsumer  $D_1$  of  $C_1$  and a subsumee  $D_2$  of  $C_2$  are. These missing entailments become the constructible  $\mathcal{H}$ . The condition  $C_{\mathcal{B},t} \not\sqsubseteq_{\sqcap} C_{\mathcal{A},t}$  is a way to write that  $C_{\mathcal{A},t} \sqsubseteq C_{\mathcal{B},t}$  is not a tautology, which can be tested by subset inclusion.

The formal proof of this result is detailed in the technical report [16]. We sketch it briefly here. To start, we link the subsumers of  $C_1$  with  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$ . This is done at the semantics level: We show that all Herbrand models of  $\Phi$ , i.e., models built on the symbols in  $\Phi$ , are also models of  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$ , that is itself such a model. Then we show that  $C_1(\mathbf{sk}_0)$  as well as the formulas corresponding to the subsumers of  $C_1$  in our translation are satisfied by all Herbrand models. This follows from the fact that  $\Phi$  is in fact a set of Horn clauses. Next, we show, using a similar technique, how duplicate negative ground implicates, not necessarily prime, relate to subsumees of  $C_2$ , with the restriction that there must exist a weak homomorphism from a description tree of a subsumer of  $C_1$  to a description tree of the considered subsumee of  $C_2$ . Thus,  $\mathcal{H}$  provides the missing CIs that will turn the weak homomorphism into a  $(\mathcal{T} \cup \mathcal{H})$ -homomorphism. Then, we establish an equivalence between the  $\sqsubseteq_{\sqcap}$ -minimality of the subsumee of  $C_2$  and the primality of the corresponding negative implicate. Packability is the last aspect we deal with, whose use is purely limited to the reconstruction. It holds because  $\mathcal{A}$  contains all  $A(t) \in \mathcal{PT}_{\Sigma}^{g+}(\Phi)$  for all terms  $t$  occurring in  $\mathcal{B}$ .

*Example 12.* Consider the abduction problem  $\langle \mathcal{T}_a, \Sigma, \alpha_a \rangle$  where  $\Sigma$  contains all concepts from  $\mathcal{T}_a$ . For the translation  $\Phi$  of this problem, we have

$$\begin{aligned} \mathcal{PT}_{\Sigma}^{g+}(\Phi) &= \{ \text{Professor}(\mathbf{sk}_0), \text{Doctor}(\mathbf{sk}_0), \text{Chair}(\mathbf{sk}_1(\mathbf{sk}_0)), \text{PhD}(\mathbf{sk}_2(\mathbf{sk}_0)) \} \\ \mathcal{PT}_{\Sigma}^{g-}(\Phi) &= \{ \neg \text{Researcher}^-(\mathbf{sk}_0), \\ &\quad \neg \text{ResearchPosition}^-(\mathbf{sk}_1(\mathbf{sk}_0)) \vee \neg \text{Diploma}^-(\mathbf{sk}_2(\mathbf{sk}_0)) \} \end{aligned}$$

where  $\mathbf{sk}_1$  is the Skolem function introduced for  $\text{Professor} \sqsubseteq \exists \text{employment.Chair}$  and  $\mathbf{sk}_2$  is introduced for  $\text{Doctor} \sqsubseteq \exists \text{qualification.PhD}$ . This leads to two constructible solutions:  $\{\text{Professor} \sqcap \text{Doctor} \sqsubseteq \text{Researcher}\}$  and  $\mathcal{H}_{a1}$ , that are both

packed connection-minimal hypotheses if  $\Sigma = \mathbf{N}_C$ . Another example is presented in full details in the technical report [16].

**Termination.** If  $\mathcal{T}$  contains cycles, there can be infinitely many prime implicates. For example, for  $\mathcal{T} = \{C_1 \sqsubseteq A, A \sqsubseteq \exists r.A, \exists r.B \sqsubseteq B, B \sqsubseteq C_2\}$  both the positive and negative ground prime implicates of  $\Phi$  are unbounded even though the set of constructible hypotheses is finite (as it is for any abduction problem):

$$\begin{aligned}\mathcal{PI}_{\Sigma}^{g+}(\Phi) &= \{C_1(\mathbf{sk}_0), A(\mathbf{sk}_0), A(\mathbf{sk}(\mathbf{sk}_0)), A(\mathbf{sk}(\mathbf{sk}(\mathbf{sk}_0))), \dots\}, \\ \mathcal{PI}_{\Sigma}^{g-}(\Phi) &= \{\neg C_2^-(\mathbf{sk}_0), \neg B^-(\mathbf{sk}_0), \neg B^-(\mathbf{sk}(\mathbf{sk}_0)), \dots\}.\end{aligned}$$

To find all constructible hypotheses of an abduction problem, an approach that simply computes all prime implicates of  $\Phi$ , e.g., using the standard resolution calculus, will never terminate on cyclic problems. However, if we look only for subset-minimal constructible hypotheses, termination can be achieved for cyclic and non-cyclic problems alike, because it is possible to construct all such hypotheses from prime implicates that have a polynomially bounded term depth, as shown below. To obtain this bound, we consider resolution derivations of the ground prime implicates and we show that they can be done under some restrictions that imply this bound.

Before performing resolution, we compute the *presaturation*  $\Phi_p$  of the set of clauses  $\Phi$ , defined as

$$\Phi_p = \Phi \cup \{\neg A(x) \vee B(x) \mid \Phi \models \neg A(x) \vee B(x)\}$$

where  $A$  and  $B$  are either both original or both duplicate atomic concepts. The presaturation can be efficiently computed before the translation, using a modern  $\mathcal{EL}$  reasoner such as ELK [23], which is highly optimized towards the computation of all entailments of the form  $A \sqsubseteq B$ . While the presaturation computes nothing a resolution procedure could not derive, it is what allows us to bind the maximal depth of terms in inferences to that in prime implicates. If  $\Phi_p$  is presaturated, we do not need to perform inferences that produce Skolem terms of a higher nesting depth than what is needed for the prime implicates.

Starting from the presaturated set  $\Phi_p$ , we can show that all the relevant prime implicates can be computed if we restrict all inferences to those where

- R1** at least one premise contains a ground term,
- R2** the resolvent contains at most one variable, and
- R3** every literal in the resolvent contains Skolem terms of nesting depth at most  $n \times m$ , where  $n$  is the number of atomic concepts in  $\Phi$ , and  $m$  is the number of occurrences of existential role restrictions in  $\mathcal{T}$ .

The first restriction turns the derivation of  $\mathcal{PI}_{\Sigma}^{g+}(\Phi)$  and  $\mathcal{PI}_{\Sigma}^{g-}(\Phi)$  into an SOS-resolution derivation [18] with set of support  $\{C_1(\mathbf{sk}_0), C_2^-(\mathbf{sk}_0)\}$ , i.e., the only two clauses with ground terms in  $\Phi$ . This restriction is a straightforward consequence of our interest in computing only ground implicates, and of the fact that the non-ground clauses in  $\Phi$  cannot entail the empty clause since every  $\mathcal{EL}$  TBox is consistent. The other restrictions are consequences of the following theorems, whose proofs are available in the technical report [16].

**Theorem 13.** *Given an abduction problem and its translation  $\Phi$ , every constructible hypothesis can be built from prime implicates that are inferred under restriction 4.*

In fact, for  $\mathcal{PT}_{\Sigma}^{g+}(\Phi)$  it is even possible to restrict inferences to generating only ground resolvents, as can be seen in the proof of Theorem 13, that directly looks at the kinds of clauses that are derivable by resolution from  $\Phi$ .

**Theorem 14.** *Given an abduction problem and its translation  $\Phi$ , every subset-minimal constructible hypothesis can be built from prime implicates that have a nesting depth of at most  $n \times m$ , where  $n$  is the number of atomic concepts in  $\Phi$ , and  $m$  is the number of occurrences of existential role restrictions in  $\mathcal{T}$ .*

The proof of Theorem 14 is based on a structure called a *solution tree*, which resembles a description tree, but with multiple labeling functions. It assigns to each node a Skolem term, a set of atomic concepts called *positive label*, and a single atomic concept called *negative label*. The nodes correspond to matching partners in a constructible hypothesis: The Skolem term is the term on which we match literals. The positive label collects the atomic concepts in the positive prime implicates containing that term. The maximal anti-chains of the tree, i.e., the maximal subsets of nodes s.t. no node is the ancestor of another are such that their negative labels correspond to the literals in a derivable negative implicate. For every solution tree, the Skolem labels and negative labels of the leaves determine a negative prime implicate, and by combining the positive and negative labels of these leaves, we obtain a constructible hypothesis, called the *solution* of the tree. We show that from every solution tree with solution  $\mathcal{H}$  we can obtain a solution tree with solution  $\mathcal{H}' \subseteq \mathcal{H}$  s.t. on no path, there are two nodes that agree both on the head of their Skolem labeling and on the negative label. Furthermore the number of head functions of Skolem labels is bounded by the total number  $n$  of Skolem functions, while the number of distinct negative labels is bounded by the number  $m$  of atomic concepts, bounding the depth of the solution tree for  $\mathcal{H}'$  at  $n \times m$ . This justifies the bound in Theorem 14. This bound is rather loose. For the academia example, it is equal to  $22 \times 6 = 132$ .

## 5 Implementation

We implemented our method to compute all subset-minimal constructible hypotheses in the tool CAPI.<sup>3</sup> To compute the prime implicates, we used SPASS [34], a first-order theorem prover that includes resolution among other calculi. We implemented everything before and after the prime implicate computation in Java, including the parsing of ontologies, preprocessing (detailed below), clausification of the abduction problems, translation to SPASS input, as well as the parsing and processing of the output of SPASS to build the constructible hypotheses and filter out the non-subset-minimal ones. On the Java side, we used the OWL API for all DL-related functionalities [20], and the  $\mathcal{EL}$  reasoner ELK for computing the presaturations [23].

<sup>3</sup> available under <https://lat.inf.tu-dresden.de/~koopmann/CAPI>.

*Preprocessing.* Since realistic TBoxes can be too large to be processed by SPASS, we replace the background knowledge in the abduction problem by a subset of axioms relevant to the abduction problem. Specifically, we replace the abduction problem  $(\mathcal{T}, \Sigma, C_1 \sqsubseteq C_2)$  by the abduction problem  $(\mathcal{M}_{C_1}^\perp \cup \mathcal{M}_{C_2}^\top, \Sigma, C_1 \sqsubseteq C_2)$ , where  $\mathcal{M}_{C_1}^\perp$  is the  $\perp$ -module of  $\mathcal{T}$  for the signature of  $C_1$ , and  $\mathcal{M}_{C_2}^\top$  is the  $\top$ -module of  $\mathcal{T}$  for the signature of  $C_2$  [15]. Those notions are explained in the technical report [16]. Their relevant properties are that  $\mathcal{M}_{C_1}^\perp$  is a subset of  $\mathcal{T}$  s.t.  $\mathcal{M}_{C_1}^\perp \models C_1 \sqsubseteq D$  iff  $\mathcal{T} \models C_1 \sqsubseteq D$  for all concepts  $D$ , while  $\mathcal{M}_{C_2}^\top$  is a subset of  $\mathcal{T}$  that ensures  $\mathcal{M}_{C_2}^\top \models D \sqsubseteq C_2$  iff  $\mathcal{T} \models D \sqsubseteq C_2$  for all concepts  $D$ . It immediately follows that every connection-minimal hypothesis for the original problem  $(\mathcal{T}, \Sigma, C_1 \sqsubseteq C_2)$  is also a connection-minimal hypothesis for  $(\mathcal{M}_{C_1}^\perp \cup \mathcal{M}_{C_2}^\top, \Sigma, C_1 \sqsubseteq C_2)$ . For the presaturation, we compute with ELK all CIs of the form  $A \sqsubseteq B$  s.t.  $\mathcal{M}_{C_1}^\perp \cup \mathcal{M}_{C_2}^\top \models A \sqsubseteq B$ .

*Prime implicates generation.* We rely on a slightly modified version of SPASS v3.9 to compute all ground prime implicates. In particular, we added the possibility to limit the number of variables allowed in the resolvents to enforce **R2**. For each of the restrictions **R1–R3** there is a corresponding flag (or set of flags) that is passed to SPASS as an argument.

*Recombination.* The construction of hypotheses from the prime implicates found in the previous stage starts with a straightforward process of matching negative prime implicates with a set of positive ones based on their Skolem terms. It is followed by subset minimality tests to discard non-subset-minimal hypotheses, since, with the bound we enforce, there is no guarantee that these are valid constructible hypotheses because the negative ground implicates they are built upon may not be prime. If SPASS terminates due to a timeout instead of reaching the bound, then it is possible that some subset-minimal constructible hypotheses are not found, and thus, some non-constructible hypotheses may be kept. Note that these are in any case solutions to the abduction problem.

## 6 Experiments

There is no benchmark suite dedicated to TBox abduction in  $\mathcal{EL}$ , so we created our own, using realistic ontologies from the bio-medical domain. For this, we used ontologies from the 2017 snapshot of Bioportal [27]. We restricted each ontology to its  $\mathcal{EL}$  fragment by filtering out unsupported axioms, where we replaced domain axioms and n-ary equivalence axioms in the usual way [2]. Note that, even if the ontology contains more expressive axioms, an  $\mathcal{EL}$  hypothesis is still useful if found. From the resulting set of TBoxes, we selected those containing at least 1 and at most 50,000 axioms, resulting in a set of 387  $\mathcal{EL}$  TBoxes. Precisely, they contained between 2 and 46,429 axioms, for an average of 3,039 and a median of 569. Towards obtaining realistic benchmarks, we created three different categories of abduction problems for each ontology  $\mathcal{T}$ , where in each case, we used the signature of the entire ontology for  $\Sigma$ .

- Problems in **ORIGIN** use  $\mathcal{T}$  as background knowledge, and as observation a randomly chosen  $A \sqsubseteq B$  s.t.  $A$  and  $B$  are in the signature of  $\mathcal{T}$  and  $\mathcal{T} \not\models A \sqsubseteq B$ . This covers the basic requirements of an abduction problem, but has the disadvantage that  $A$  and  $B$  can be completely unrelated in  $\mathcal{T}$ .
- Problems in **JUSTIF** contain as observation a randomly selected CI  $\alpha$  s.t., for the original TBox,  $\mathcal{T} \models \alpha$  and  $\alpha \notin \mathcal{T}$ . The background knowledge used is a *justification for  $\alpha$  in  $\mathcal{T}$*  [32], that is, a minimal subset  $\mathcal{I} \subseteq \mathcal{T}$  s.t.  $\mathcal{I} \models \alpha$ , from which a randomly selected axiom is removed. The TBox is thus a smaller set of axioms extracted from a real ontology for which we know there is a way of producing the required entailment without adding it explicitly. Justifications were computed using functionalities of the OWL API and ELK.
- Problems in **REPAIR** contain as observation a randomly selected CI  $\alpha$  s.t.  $\mathcal{T} \models \alpha$ , and as background knowledge a *repair for  $\alpha$  in  $\mathcal{T}$* , which is a maximal subset  $\mathcal{R} \subseteq \mathcal{T}$  s.t.  $\mathcal{R} \not\models \alpha$ . Repairs were computed using a justification-based algorithm [32] with justifications computed as for **JUSTIF**. This usually resulted in much larger TBoxes, where more axioms would be needed to establish the entailment.

All experiments were run on Debian Linux (Intel Core i5-4590, 3.30 GHz, 23 GB Java heap size). The code and scripts used in the experiments are available online [17]. The three phases of the method (see Fig. 2) were each assigned a hard time limit of 90 s.

For each ontology, we attempted to create and translate 5 abduction problems of each category. This failed on some ontologies because either there was no corresponding entailment (25/28/25 failures out of the 387 ontologies for **ORIGIN**/**JUSTIF**/**REPAIR**), there was a timeout during the translation (5/5/5 failures for **ORIGIN**/**JUSTIF**/**REPAIR**), or because the computation of justifications caused an exception (-/2/0 failures for **ORIGIN**/**JUSTIF**/**REPAIR**). The final number of abduction problems for each category is in the first column of Table 1.

We then attempted to compute prime implicates for these benchmarks using SPASS. In addition to the hard time limit, we gave a soft time limit of 30 s to SPASS, after which it should stop exploring the search space and return the implicates already found. In Table 1 we show, for each category, the percentage of problems on which SPASS succeeded in computing a non-empty set of clauses (Success) and the percentage of problems on which SPASS terminated within the time limit, where all solutions are computed (Compl.). The high number of CIs in the background knowledge explains most of the cases where SPASS reached the soft time limit. In a lot of these cases, the bound on the term depth goes into the billion, rendering it useless in practice. However, the “Compl.” column shows that the bound is reached before the soft time limit in most cases.

The reconstruction never reached the hard time limit. We measured the median, average and maximal number of solutions found ( $\#\mathcal{H}$ ), size of solutions in number of CIs ( $|\mathcal{H}|$ ), size of CIs from solutions in number of atomic concepts ( $|\alpha|$ ), and SPASS runtime (time, in seconds), all reported in Table 1. Except for the simple **JUSTIF** problems, the number of solutions may become very large. At the same time, solutions always contain very few axioms (never

**Table 1.** Evaluation results.

	#Probl.	Success	Compl.	Median / avg / max			
				# $\mathcal{H}$	$ \mathcal{H} $	$ \alpha $	Time (s.)
ORIGIN	1,925	94.7%	61.3%	1/8.51/1850	1/1.00/2	6/7.48/91	0.2/12.4/43.8
JUSTIF	1,803	100.0%	97.2%	1/1.50/5	1/1/1	2/4.21/32	0.2/1.1/34.1
REPAIR	1,805	92.9%	57.0%	43/228.05/6317	1/1.00/2	5/5.09/49	0.6/13.6/59.9

more than 3), though the axioms become large too. We also noticed that highly nested Skolem terms rarely lead to more hypotheses being found: 8/1/15 for ORIGIN/JUSTIF/REPAIR, and the largest nesting depth used was: 3/1/2 for ORIGIN/JUSTIF/REPAIR. This hints at the fact that longer time limits would not have produced more solutions, and motivates future research into redundancy criteria to stop derivations (much) earlier.

## 7 Conclusion

We have introduced connection-minimal TBox abduction for  $\mathcal{EL}$  which finds parsimonious hypotheses, ruling out the ones that entail the observation in an arbitrary fashion. We have established a formal link between the generation of connection-minimal hypotheses in  $\mathcal{EL}$  and the generation of prime implicates of a translation  $\Phi$  of the problem to first-order logic. In addition to obtaining these theoretical results, we developed a prototype for the computation of subset-minimal constructible hypotheses, a subclass of connection-minimal hypotheses that is easy to construct from the prime implicates of  $\Phi$ . Our prototype uses the SPASS theorem prover as an SOS-resolution engine to generate the needed implicates. We tested this tool on a set of realistic medical ontologies, and the results indicate that the cost of computing connection-minimal hypotheses is high but not prohibitive.

We see several ways to improve our technique. The bound we computed to ensure termination could be advantageously replaced by a redundancy criterion discarding irrelevant implicates long before it is reached, thus greatly speeding computation in SPASS. We believe it should also be possible to further constrain inferences, e.g., to have them produce ground clauses only, or to generate the prime implicates with terms of increasing depth in a controlled incremental way instead of enforcing the soft time limit, but these two ideas remain to be proved feasible. As an alternative to using prime implicates, one may investigate direct method for computing connection-minimal hypotheses in  $\mathcal{EL}$ .

The theoretical worst-case complexity of connection-minimal abduction is another open question. Our method only gives a very high upper bound: by bounding only the nesting dept of Skolem terms polynomially as we did with Theorem 13, we may still permit clauses with exponentially many literals, and thus double exponentially many clauses in the worst case, which would give us an 2EXPTIME upper bound to the problem of computing all subset-minimal constructible hypotheses. Using structure-sharing and guessing, it is likely possible



to get a lower bound. We have not looked yet at lower bounds for the complexity either.

While this work focuses on abduction problems where the observation is a CI, we believe that our technique can be generalised to knowledge that also contains ground facts (ABoxes), and to observations that are of the form of conjunctive queries on the ABoxes in such knowledge bases. The motivation for such an extension is to understand why a particular query does not return any results, and to compute a set of TBox axioms that fix this problem. Since our translation already transforms the observation into ground facts, it should be possible to extend it to this setting. We would also like to generalize TBox abduction by finding a reasonable way to allow role restrictions in the hypotheses, and to extend connection-minimality to more expressive DLs such as  $\mathcal{ALC}$ .

**Acknowledgments.** This work was supported by the Deutsche Forschungsgemeinschaft (DFG), Grant 389792660 within TRR 248.

## References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the  $\mathcal{EL}$  envelope. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, 30 July - 5 August 2005, pp. 364–369. Professional Book Center (2005). <http://ijcai.org/Proceedings/05/Papers/0372.pdf>
2. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017). <https://doi.org/10.1017/9781139025355>
3. Baader, F., Küsters, R., Molitor, R.: Computing least common subsumers in description logics with existential restrictions. In: Proceedings of IJCAI 1999, pp. 96–103. Morgan Kaufmann (1999)
4. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 19–99. Elsevier and MIT Press, Cambridge (2001). <https://doi.org/10.1016/b978-044450813-3/50004-7>
5. Bauer, J., Sattler, U., Parsia, B.: Explaining by example: model exploration for ontology comprehension. In: Grau, B.C., Horrocks, I., Motik, B., Sattler, U. (eds.) Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, 27–30 July 2009. CEUR Workshop Proceedings, vol. 477. CEUR-WS.org (2009). [http://ceur-ws.org/Vol-477/paper\\_37.pdf](http://ceur-ws.org/Vol-477/paper_37.pdf)
6. Biennu, M.: Complexity of abduction in the  $\mathcal{EL}$  family of lightweight description logics. In: Proceedings of KR 2008, pp. 220–230. AAAI Press (2008), <http://www.aaai.org/Library/KR/2008/kr08-022.php>
7. Calvanese, D., Ortiz, M., Simkus, M., Stefanoni, G.: Reasoning about explanations for negative query answers in DL-Lite. J. Artif. Intell. Res. **48**, 635–669 (2013). <https://doi.org/10.1613/jair.3870>
8. Ceylan, İ.İ., Lukasiewicz, T., Malizia, E., Molinaro, C., Vaicnavicius, A.: Explanations for negative query answers under existential rules. In: Calvanese, D., Erdem, E., Thielscher, M. (eds.) Proceedings of KR 2020, pp. 223–232. AAAI Press (2020). <https://doi.org/10.24963/kr.2020/23>

9. Del-Pinto, W., Schmidt, R.A.: ABox abduction via forgetting in  $\mathcal{ALC}$ . In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, pp. 2768–2775. AAAI Press (2019). <https://doi.org/10.1609/aaai.v33i01.33012768>
10. Du, J., Qi, G., Shen, Y., Pan, J.Z.: Towards practical ABox abduction in large description logic ontologies. *Int. J. Semantic Web Inf. Syst.* **8**(2), 1–33 (2012). <https://doi.org/10.4018/jswis.2012040101>
11. Du, J., Wan, H., Ma, H.: Practical TBox abduction based on justification patterns. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, pp. 1100–1106 (2017). <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14402>
12. Du, J., Wang, K., Shen, Y.: A tractable approach to ABox abduction over description logic ontologies. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, pp. 1034–1040. AAAI Press (2014). <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8191>
13. Echenim, M., Peltier, N., Sellami, Y.: A generic framework for implicate generation modulo theories. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 279–294. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-94205-6\\_19](https://doi.org/10.1007/978-3-319-94205-6_19)
14. Elsenbroich, C., Kutz, O., Sattler, U.: A case for abductive reasoning over ontologies. In: Proceedings of the OWLED’06 Workshop on OWL: Experiences and Directions (2006). <http://ceur-ws.org/Vol-216/submission.25.pdf>
15. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: theory and practice. *J. Artif. Intell. Res.* **31**, 273–318 (2008). <https://doi.org/10.1613/jair.2375>
16. Haifani, F., Koopmann, P., Tourret, S., Weidenbach, C.: Connection-minimal abduction in  $\mathcal{EL}$  via translation to FOL - technical report (2022). <https://doi.org/10.48550/ARXIV.2205.08449>, <https://arxiv.org/abs/2205.08449>
17. Haifani, F., Koopmann, P., Tourret, S., Weidenbach, C.: Experiment data for the paper Connection-minimal Abduction in EL via translation to FOL, May 2022. <https://doi.org/10.5281/zenodo.6563656>
18. Haifani, F., Tourret, S., Weidenbach, C.: Generalized completeness for SOS resolution and its application to a new notion of relevance. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 327–343. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_19](https://doi.org/10.1007/978-3-030-79876-5_19)
19. Halland, K., Britz, K.: ABox abduction in  $\mathcal{ALC}$  using a DL tableau. In: 2012 South African Institute of Computer Scientists and Information Technologists Conference, SAICSIT ’12, pp. 51–58 (2012). <https://doi.org/10.1145/2389836.2389843>
20. Horridge, M., Bechhofer, S.: The OWL API: a java API for OWL ontologies. *Semant. Web* **2**(1), 11–21 (2011). <https://doi.org/10.3233/SW-2011-0025>
21. Horridge, M., Parsia, B., Sattler, U.: Explanation of OWL entailments in protege 4. In: Bizer, C., Joshi, A. (eds.) Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, 28 October 2008. CEUR Workshop Proceedings, vol. 401. CEUR-WS.org (2008). [http://ceur-ws.org/Vol-401/iswc2008pd\\_submission.47.pdf](http://ceur-ws.org/Vol-401/iswc2008pd_submission.47.pdf)
22. Kazakov, Y., Klinov, P., Stupnikov, A.: Towards reusable explanation services in protege. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, 18–21 July 2017. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017). <http://ceur-ws.org/Vol-1879/paper31.pdf>

23. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK - from polynomial procedures to efficient reasoning with  $\mathcal{EL}$  ontologies. *J. Autom. Reason.* **53**(1), 1–61 (2014). <https://doi.org/10.1007/s10817-013-9296-3>
24. Klarman, S., Endriss, U., Schlobach, S.: ABox abduction in the description logic  $\mathcal{ALC}$ . *J. Autom. Reason.* **46**(1), 43–80 (2011). <https://doi.org/10.1007/s10817-010-9168-z>
25. Koopmann, P.: Signature-based abduction with fresh individuals and complex concepts for description logics. In: Zhou, Z. (ed.) *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event/Montreal, Canada, 19–27 August 2021*, pp. 1929–1935 (2021). <https://doi.org/10.24963/ijcai.2021/266>
26. Koopmann, P., Del-Pinto, W., Tourret, S., Schmidt, R.A.: Signature-based abduction for expressive description logics. In: Calvanese, D., Erdem, E., Thielscher, M. (eds.) *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020*, pp. 592–602. AAAI Press (2020). <https://doi.org/10.24963/kr.2020/59>
27. Matentzoglou, N., Parsia, B.: Biportal snapshot 30.03.2017 (2017). <https://doi.org/10.5281/zenodo.439510>
28. Nabeshima, H., Iwanuma, K., Inoue, K., Ray, O.: SOLAR: an automated deduction system for consequence finding. *AI Commun.* **23**(2–3), 183–203 (2010). <https://doi.org/10.3233/AIC-2010-0465>
29. Parsia, B., Matentzoglou, N., Gonçalves, R.S., Glimm, B., Steigmiller, A.: The owl reasoner evaluation (ORE) 2015 competition report. *J. Autom. Reason.* **59**(4), 455–482 (2017). <https://doi.org/10.1007/s10817-017-9406-8>
30. Pukancová, J., Homola, M.: Tableau-based ABox abduction for the  $\mathcal{ALCHO}$  description logic. In: *Proceedings of the 30th International Workshop on Description Logics* (2017). <http://ceur-ws.org/Vol-1879/paper11.pdf>
31. Pukancová, J., Homola, M.: The AAA Abox abduction solver. *KI - Künstliche Intell.* **34**(4), 517–522 (2020). <https://doi.org/10.1007/s13218-020-00685-4>
32. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pp. 355–362. Morgan Kaufmann, Acapulco, Mexico (2003). <http://ijcai.org/Proceedings/03/Papers/053.pdf>
33. Wei-Kleiner, F., Dragisic, Z., Lambrix, P.: Abduction framework for repairing incomplete  $\mathcal{EL}$  ontologies: complexity results and algorithms. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pp. 1120–1127. AAAI Press (2014). <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8239>
34. Weidenbach, C., Schmidt, R.A., Hillenbrand, T., Rusev, R., Topic, D.: System description: SPASS version 3.0. In: Pfenning, F. (ed.) *CADE 2007. LNCS (LNAI)*, vol. 4603, pp. 514–520. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73595-3\\_38](https://doi.org/10.1007/978-3-540-73595-3_38)
35. Wos, L., Robinson, G., Carson, D.: Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM* **12**(4), 536–541 (1965)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Semantic Relevance

Fajar Haifani<sup>1,2</sup>  and Christoph Weidenbach<sup>1</sup>  

<sup>1</sup> Max Planck Institute for Informatics, Saarland Informatics Campus,  
Saarbrücken, Germany

`{f.haifani, weidenbach}@mpi-inf.mpg.de`

<sup>2</sup> Graduate School of Computer Science, Saarbrücken, Germany

**Abstract.** A clause  $C$  is syntactically relevant in some clause set  $N$ , if it occurs in every refutation of  $N$ . A clause  $C$  is syntactically semi-relevant, if it occurs in some refutation of  $N$ . While syntactic relevance coincides with satisfiability (if  $C$  is syntactically relevant then  $N \setminus \{C\}$  is satisfiable), the semantic counterpart for syntactic semi-relevance was not known so far. Using the new notion of a *conflict literal* we show that for independent clause sets  $N$  a clause  $C$  is syntactically semi-relevant in the clause set  $N$  if and only if it adds to the number of conflict literals in  $N$ . A clause set is independent, if no clause out of the clause set is the consequence of different clauses from the clause set.

Furthermore, we relate the notion of relevance to that of a minimally unsatisfiable subset (MUS) of some independent clause set  $N$ . In propositional logic, a clause  $C$  is relevant if it occurs in all MUSes of some clause set  $N$  and semi-relevant if it occurs in some MUS. For first-order logic the characterization needs to be refined with respect to ground instances of  $N$  and  $C$ .

## 1 Introduction

In our previous work [11], we introduced a notion of syntactic relevance based on refutations while at the same time generalized the completeness result for resolution by the set-of-support strategy (SOS) [28, 33] as its test. Our notion of syntactic relevance is useful for explaining why a set of clauses is unsatisfiable. In this paper, we introduce a semantic counterpart of syntactic relevance that sheds further light on the relationship between a clause out of a clause set and the potential refutations of this clause set. In the following Sect. 1.1, we first recall syntactic relevance along with an example and then proceeds to explain it in terms of our new semantic relevance in the later Sect. 1.2.

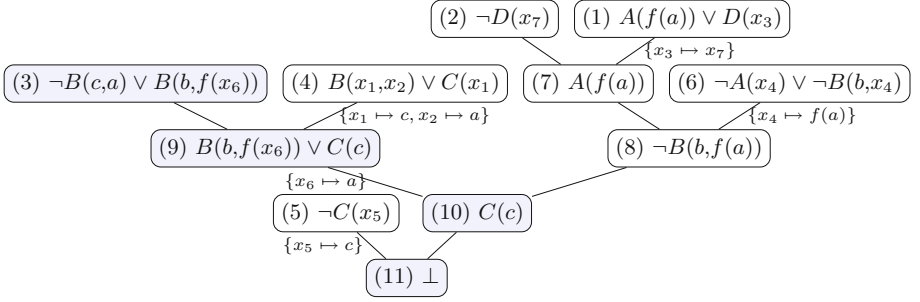
### 1.1 Syntactic Relevance

Given an unsatisfiable set of clauses  $N$ ,  $C \in N$  is *syntactically relevant* if it occurs in all refutations, it is *syntactically semi-relevant* if it occurs in some refutation, otherwise it is called *syntactically irrelevant*. The clause-based notion of relevance is useful in relating the contribution of a clause to refutation (goal conjecture).

This has in particular been shown in the context of product scenarios built out of construction kits as they are used in the car industry [8,32].

For an illustration of our previous notions and results we now consider the following unsatisfiable first-order clause set  $N$  where Fig. 1 presents a refutation of  $N$ .

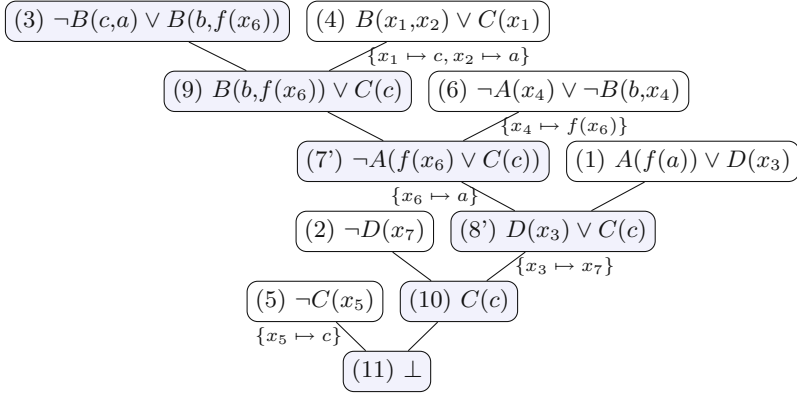
$$N = \{(1) A(f(a)) \vee D(x_3), \\ (2) \neg D(x_7), \\ (3) \neg B(c, a) \vee B(b, f(x_6)), \\ (4) B(x_1, x_2) \vee C(x_1), \\ (5) \neg C(x_5), \\ (6) \neg A(x_4) \vee \neg B(b, x_4)\}$$



**Fig. 1.** A refutation of  $N$  in tree representation

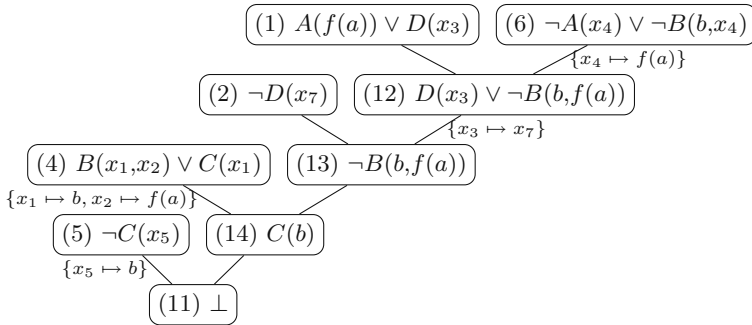
In essence, inferences in an SOS refutation always involve at least one clause in the SOS and put the resulting clause back in it. So, this refutation is not an SOS refutation from the syntactically semi-relevant clause (3)  $\neg B(c, a) \vee B(b, f(x_6))$ , because only the shaded part represents an SOS refutation starting with this clause. More specifically, there are two inferences ended in (8)  $\neg B(b, f(a))$  which violates the condition for an SOS refutation. Nevertheless, it can be transformed into an SOS refutation where the clause (3)  $\neg B(c, a) \vee B(b, f(x_6))$  is in the SOS [11], Fig. 2. Please note that  $N \setminus \{(3) \neg B(c, a) \vee B(b, f(x_6))\}$  is still unsatisfiable and classical SOS completeness [33] is not sufficient to guarantee the existence of a refutation with SOS  $\{(3) \neg B(c, a) \vee B(b, f(x_6))\}$  [11].

In addition,  $N \setminus \{(3) \neg B(c, a) \vee B(b, f(x_6))\}$  is also a *minimally unsatisfiable subset* (MUS), where Fig. 3 presents a respective refutation. A MUS is an unsatisfiable clause set such that removing a clause from this set would render it satisfiable. Consequently, a MUS-based defined notion of semi-relevance on the level of the original first-order clauses is not sufficient here. The clause



**Fig. 2.** Semi-relevant clause  $(3) \neg B(c, a) \vee B(b, f(x_6))$  in SOS

$(3) \neg B(c, a) \vee B(b, f(x_6))$  should not be disregarded, because it leads to a different grounding of the clauses. For example, in the refutation of Fig. 2 clause  $(5) \neg C(x_5)$  is necessarily instantiated with  $\{x_5 \mapsto c\}$  where in the refutation of Fig. 3 it is necessarily instantiated with  $\{x_5 \mapsto b\}$ . Therefore, the two refutations are different and clause  $(3) \neg B(c, a) \vee B(b, f(x_6))$  should be considered semi-relevant. Nevertheless, in propositional logic it is sufficient to consider MUSes to explain unsatisfiability on the original clause level, Lemma 18.



**Fig. 3.** A refutation of  $N$  without  $(3) \neg B(c, a) \vee B(b, f(x_6))$

## 1.2 Semantic Relevance

We now illustrate how our new notion of relevance works on the previous example. First, different from the other works, we propose a way of characterizing semantic relevance by using our novel concept of a *conflict literal*. A ground

literal  $L$  is a conflict literal in a clause set  $N$  if there are some satisfiable sets of instances  $N_1$  and  $N_2$  from  $N$  s.t.  $N_1 \models L$  and  $N_2 \models \text{comp}(L)$ . On the one hand, explaining an unsatisfiable clause set as the absence of a model (as it is usually defined) is not that helpful since an absence means there is nothing to discuss in the first place. On the other hand, the contribution of a clause to unsatisfiability of a clause set can only partially be explained using the concept of a MUS which we have discussed before. A conflict literal provides a middle ground to explain the contribution of a clause to unsatisfiability between the absence of a model and MUSes. It also better reflects our intuition that there is a contradiction (in the form of two implied simple facts that cannot be both true at the same time) in an unsatisfiable set of clauses.

From Fig. 1, we can already see that  $C(c)$  and its complement  $\neg C(c)$  are conflict literals because

$$\begin{aligned} N \setminus \{\neg C(x)\} &\models C(c) \\ \neg C(x) &\models \neg C(c) \end{aligned}$$

Also, in addition to that  $\{\neg C(x)\}$  is trivially satisfiable,  $N \setminus \{\neg C(x)\}$  is also satisfiable. Based on the refutation in Fig. 3,  $\neg C(x)$  is syntactically relevant due to  $N \setminus \{(3)\neg B(c, a) \vee B(b, f(x_6))\}$  being a MUS. We will also show that for a ground MUS any ground literal occurring in it is a conflict literal, Lemma 20. For our ongoing example it is still possible to identify the conflict literals by means of ground MUSes by looking into the refutations from Fig. 1 and Fig. 3. This leads to the following conflict literals for  $N$ , see Definition 10:

$$\begin{aligned} \text{conflict}(N) = &\{(\neg)A(f(a)), \\ &(\neg)B(b, f(a)), (\neg)B(c, a), \\ &(\neg)C(b), (\neg)C(c)\} \cup \\ &\{(\neg)D(t) \mid t \text{ is a ground term}\} \end{aligned}$$

These conflict literals can be identified by pushing the substitutions in the refutations from Fig. 1 and Fig. 3 towards the input clauses. They correspond to two first-order MUSes  $M_1$  and  $M_2$ . All the ground literals are conflict literals and all other ground conflict literals can be obtained by grounding the remaining variables.

$$\begin{aligned} M_1 = &\{(5)\neg C(c), (2)\neg D(x_7), \\ &(1)A(f(a)) \vee D(x_3), \\ &(3)\neg B(c, a) \vee B(b, f(a)), \\ &(4)B(c, a) \vee C(c), \\ &(6)\neg A(f((a))) \vee \neg B(b, f(a))\} \\ M_2 = &\{(5)\neg C(b), \\ &(4)B(b, f(a)), (2)\neg D(x_7), \\ &(1)A(f(a)) \vee D(x_3), \\ &(6)\neg A(f(a)) \vee \neg B(b, f(a))\} \end{aligned}$$



One can see that, despite  $(3)\neg B(c, a) \vee B(b, f(x_6))$  is outside of the only MUS on the first-order level, an instance of it does occur in some ground MUS, take  $M_1$  and an arbitrary grounding of  $x_3$  and  $x_7$  to the identical term  $t$ , and the conflict literal  $(\neg)B(c, a)$  depends on clause (3). Nevertheless, determining conflict literals is not so obvious in the general case since we do not necessarily know beforehand which ground terms should substitute the variables in the clauses. Moreover, there can be an infinite number of such ground MUSes of possibly unbounded size.

Based on conflict literals, here we introduce a notion of relevance that is semantic in nature, Definition 16. This will also serve as an alternative characterization to our previous refutation-based syntactic relevance. As redundant clauses, e.g., tautologies, can also be syntactically semi-relevant, we require independent clause sets for the definition of semantic relevance. A clause set is *independent*, if it does not contain clauses with instances implied by satisfiable sets of instances of different clauses out of the set. Given an unsatisfiable independent set of clauses  $N$ , a clause  $C$  is *relevant* in  $N$  if  $N$  without  $C$  has no conflict literals, it is *semi-relevant* if  $C$  is necessary to some conflict literals, and it is *irrelevant* otherwise.

Similar to our previous work, relevant clauses are the obvious ones because removing them would make our set satisfiable. On the other hand, irrelevant clauses can be freely identified once we know the semi-relevant ones. For our running example, in fact  $(3)\neg B(c, a) \vee B(b, f(x_6))$  is semi-relevant because it is necessary for the conflict literals  $(\neg)C(c)$  and  $(\neg)B(c, a)$ . More specifically, the set of conflicts for  $N \setminus \{\neg B(c, a) \vee B(b, f(x_6))\}$  does not include  $(\neg)C(c)$  and  $(\neg)B(c, a)$ :

$$\text{conflict}(N \setminus \{\neg B(c, a) \vee B(b, f(x_6))\}) = \{(\neg)A(f(a)), (\neg)B(b, f(a)), (\neg)C(b)\} \uplus \{(\neg)D(t) \mid t \text{ is a ground term}\}$$

These are conflict literals identifiable from  $M_2$ : Assume that the variables  $x_3$  and  $x_7$  in  $M_2$  are both grounded by an identical term  $t$ . Take some ground literal, for example,  $A(f(a)) \in \text{conflict}(N \setminus \{\neg B(c, a) \vee B(b, f(x_6))\})$ , and define

$$\begin{aligned} N_\emptyset &= \{C \in M_2 \mid A(f(a)) \notin C \text{ and } \neg A(f(a)) \notin C\} \\ &= \{(5)\neg C(b), (4)B(b, f(a)), (2)\neg D(t)\} \\ N_{A(f(a))} &= \{C \in M_2 \mid A(f(a)) \in C\} \\ &= \{(1)A(f(a)) \vee D(t)\} \\ N_{\neg A(f(a))} &= \{C \in M_2 \mid \neg A(f(a)) \in C\} \\ &= \{(6)\neg A(f(a)) \vee \neg B(b, f(a))\} \end{aligned}$$

$N_\emptyset \cup N_{A(f(a))}$  and  $N_\emptyset \cup N_{\neg A(f(a))}$  are satisfiable because of the Herbrand model  $\{B(b, f(a)), A(f(a))\}$  and  $\{B(b, f(a))\}$  respectively. In addition,

$$\begin{aligned} N_\emptyset \cup N_{A(f(a))} &\models A(f(a)) \\ N_\emptyset \cup N_{\neg A(f(a))} &\models \neg A(f(a)) \end{aligned}$$

because  $A(f(a))$  can be acquired using resolution between (1) and (2) for  $N_\emptyset \cup N_{A(f(a))}$  and  $\neg A(f(a))$  can be acquired using resolution between (4) and (6) for  $N_\emptyset \cup N_{\neg A(f(a))}$ . In a similar manner, we can show that the other ground literals are also conflict literals.

*Related Work:* Other works which aim to explain unsatisfiability mostly rely on the notion of MUSes, mainly in propositional logic [14–16, 21, 26]. The complexity of determining whether a clause set is a MUS is  $D^p$ -complete for a propositional clause set with at most three literals per clause and at most three occurrences of each propositional variable [25]. In [14], syntactically semi-relevant clauses for propositional logic are called a *plain clause set*. Using the terminology in [16], a clause  $C \in N$  is *necessary* if it occurs in all MUSes, it is *potentially necessary* if it occurs in some MUS, otherwise, it is *never necessary*. In addition, a clause is defined to be *usable* if it occurs in some refutation. This is thus similar to our syntactic notion of semi-relevance [11]: Given a clause  $C \in N$ ,  $C$  is usable if-and-only-if  $C$  is syntactically semi-relevant. It is also argued that a usable clause that is not potentially necessary is semantically superfluous. A different but related notion has also been applied for propositional abduction [7]. The notion of a MUS has also been used for explaining unsatisfiability in first-order logic [20]. There, it has been defined in a more general setting: If a set of clauses  $N$  is divided into  $N = N' \uplus N''$  with a *non-relaxable* clause set  $N'$  and *relaxable* clause set  $N''$  (which must be satisfiable), a MUS is a subset  $M$  of  $N''$  s.t.  $N' \uplus M$  is unsatisfiable but removing a clause from  $M$  would render it satisfiable. There are also some works in satisfiability modulo theory (SMT) [5, 6, 9, 35]. A deletion-based approach well-known in propositional logic has also been used for MUS extraction in SMT [9]. In [5, 6], a MUS is extracted by combining an SMT solver with an arbitrary external propositional core extractor. Another approach is to construct some graph representing the subformulas of the problem instance, recursively remove clauses in a depth-first-search manner and additionally use some heuristics to further improve the runtime [35]. For the function-free and equality-free first-order fragment, there is a “decompose-merge” approach to compute all MUSes [19, 34]. In description logic, a notion that is related to MUS is called *minimal axiom set* (MinA) usually identified by the problem of axiom pinpointing [1, 4, 13, 30]. Its computation is usually divided into two categories: black-box and white-box. A black-box approach picks some inputs, executes it using some sound and complete reasoner, and then interprets the output [13]. On the other hand, white-box approach takes some reasoner and performs an internal modification for it. In this case, Tableau is mostly used [1, 30]. In addition, the concept of a lean kernel has also been used to approximate the union of such MinA’s [27]. The way relevance is defined is similar in spirit but usually used for an entailment problem instead of unsatisfiability. The notion of syntactic semi-relevance has also been applied to description logics via a translation scheme to first-order logic [10].

The paper is organized as follows. Section 2 fixes the notations, definitions and existing results in particular from [11]. Section 3 is reserved for our new

notion of semantic relevance. Finally, we conclude our work in Sect. 4 with a discussion of our results.

## 2 Preliminaries

We assume a standard first-order language without equality over a signature  $\Sigma = (\Omega, \Pi)$  where  $\Omega$  is a non-empty set of functions symbols,  $\Pi$  a non-empty set of predicate symbols both coming with their respective fixed arities denoted by the function arity. The set of terms over an infinite set of variables  $\mathcal{X}$  is denoted by  $T(\Sigma, \mathcal{X})$ . Atoms, literals, clauses, and clause sets are defined as usual, e.g., see [24]. We identify a clause with its multiset of literals. Variables in clauses are universally quantified. Then  $N$  denotes a clause set;  $C, D$  denote clauses;  $L, K$  denote literals;  $A, B$  denote atoms;  $P, Q, R, T$  denote predicates;  $t, s$  terms;  $f, g, h$  functions;  $a, b, c, d$  constants; and  $x, y, z$  variables, all possibly indexed. The complement of a literal is denoted by the function comp. Atoms, literals, clauses, and clause sets are *ground* if they do not contain any variable.

An interpretation  $\mathcal{I}$  with a nonempty *domain* (or *universe*)  $\mathcal{U}$  assigns (i) a total function  $f^{\mathcal{I}} : \mathcal{U}^n \mapsto \mathcal{U}$  for each  $f \in \Omega$  with  $\text{arity}(f) = n$  and (ii) a relation  $P \subseteq \mathcal{U}^m$  to every predicate symbol  $P^{\mathcal{I}} \in \Pi$  with  $\text{arity}(P) = m$ . A valuation  $\beta$  is a function  $\mathcal{X} \mapsto \mathcal{U}$  where the assignment of some variable  $x$  can be modified to  $e \in \mathcal{U}$  by  $\beta[x \mapsto e]$ . It is extended to terms as  $\mathcal{I}(\beta) : T(\Sigma, \mathcal{X}) \mapsto \mathcal{U}$ . Semantic entailment  $\models$  considers variables in clauses to be universally quantified. The extension to atoms, literals, disjunctions, clauses and sets of clauses is as follows:  $\mathcal{I}(\beta)(P(t_1, \dots, t_n)) = 1$  if  $(\mathcal{I}(\beta)(t_1), \dots, \mathcal{I}(\beta)(t_n)) \in P^{\mathcal{I}}$  and 0 otherwise;  $\mathcal{I}(\beta)(\neg\phi) = 1 - \mathcal{I}(\beta)(\phi)$ ; for a disjunction  $L_1 \vee \dots \vee L_k$ ,  $\mathcal{I}(\beta)(L_1 \vee \dots \vee L_k) = \max(\mathcal{I}(\beta)(L_1), \dots, \mathcal{I}(\beta)(L_k))$ ; for a clause  $C$ ,  $\mathcal{I}(\beta)(C) = 1$  if for all valuations  $\beta = \{x_1 \mapsto e_1, \dots, x_n \mapsto e_n\}$  where the  $x_i$  are the free variables in  $C$  there is a literal  $L \in C$  such that  $\mathcal{I}(\beta)(L) = 1$ ; for a set of clauses  $N = \{C_1, \dots, C_k\}$ ,  $\mathcal{I}(\beta)(\{C_1, \dots, C_k\}) = \min(\mathcal{I}(\beta)(C_1), \dots, \mathcal{I}(\beta)(C_k))$ . A set of clauses  $N$  is *satisfiable* if there is an  $\mathcal{I}$  of  $N$  such that  $\mathcal{I}(\beta)(N) = 1$ ,  $\beta$  arbitrary, (in this case  $\mathcal{I}$  is called a *model* of  $N$ :  $\mathcal{I} \models N$ ) otherwise  $N$  is called *unsatisfiable*.

Substitutions  $\sigma, \tau$  are total mappings from variables to terms, where  $\text{dom}(\sigma) := \{x \mid x\sigma \neq x\}$  is finite and  $\text{codom}(\sigma) := \{t \mid x\sigma = t, x \in \text{dom}(\sigma)\}$ . A *renaming*  $\sigma$  is a bijective substitution. The application of substitutions is extended to literals, clauses, and sets/sequences of such objects in the usual way. If  $C' = C\sigma$  for some substitution  $\sigma$ , then  $C'$  is an *instance* of  $C$ . A *unifier*  $\sigma$  for a set of terms  $t_1, \dots, t_k$  satisfies  $t_i\sigma = t_j\sigma$  for all  $1 \leq i, j \leq k$  and it is called a *most general unifier* if for any unifier  $\sigma'$  of  $t_1, \dots, t_k$  there is a substitution  $\tau$  s.t.  $\sigma' = \sigma\tau$ . The function *mgu* denotes the *most general unifier* of two terms, atoms, literals if it exists. We assume that any mgu of two terms or literals does not introduce any fresh variables and is idempotent.

The resolution calculus consists of two inference rules: Resolution and Factoring [28, 29]. The rules operate on a state  $(N, S)$  where the initial state for a classical resolution refutation from a clause set  $N$  is  $(\emptyset, N)$  and for an SOS (Set Of Support) refutation with clause set  $N$  and initial SOS clause set  $S$  the

initial state is  $(N, S)$ . We describe the rules in the form of abstract rewrite rules operating on states  $(N, S)$ . As usual we assume for the resolution rule that the involved clauses are variable disjoint. This can always be achieved by applying renamings into fresh variables.

**Resolution**  $(N, S \uplus \{C \vee K\}) \Rightarrow_{\text{RES}} (N, S \cup \{C \vee K, (D \vee C)\sigma\})$   
provided  $(D \vee L) \in (N \cup S)$  and  $\sigma = \text{mgu}(L, \text{comp}(K))$

**Factoring**  $(N, S \uplus \{C \vee L \vee K\}) \Rightarrow_{\text{RES}} (N, S \cup \{C \vee L \vee K\} \cup \{(C \vee L)\sigma\})$   
provided  $\sigma = \text{mgu}(L, K)$

The clause  $(D \vee C)\sigma$  is the result of a *Resolution inference* between its parents and called a *resolvent*. The clause  $(C \vee L)\sigma$  is the result of a *Factoring inference* of its parent and called a *factor*. A sequence of rule applications  $(N, S) \Rightarrow_{\text{RES}}^* (N, S')$  is called a *resolution derivation*. It is called an *SOS resolution derivation* if  $N \neq \emptyset$ . In case  $\perp \in S'$  it is called a *(SOS) resolution refutation*. If for two clauses  $C, D$  there exists a substitution  $\sigma$  such that  $C\sigma \subseteq D$ , then we say that  $C$  *subsumes*  $D$ . In this case  $C \models D$ .

**Theorem 1 (Soundness and Refutational Completeness of (SOS) Resolution [11, 28, 33]).** *Resolution is sound and refutationally complete [28]. If for some clause set  $N$  and initial SOS  $S$ ,  $N$  is satisfiable and  $N \cup S$  is unsatisfiable, then there is a (SOS) resolution derivation of  $\perp$  from  $(N, S)$  [33]. If for some clause set  $N$  and clause  $C \in N$  there exists a resolution refutation from  $N$  using  $C$ , then there is an SOS derivation of  $\perp$  from  $(N \setminus \{C\}, \{C\})$  [11].*

Please note that the recent SOS completeness result of [11] generalizes the classical SOS completeness result by [33].

**Theorem 2 (Deductive Completeness of Resolution [17, 22]).** *Given a set of clauses  $N$  and a clause  $D$ , if  $N \models D$ , then there is a resolution derivation of some clause  $C$  from  $(\emptyset, N)$  such that  $C$  subsumes  $D$ .*

For deductions we require every clause to be used exactly once, so deductions always have a tree form.

**Definition 3 (Deduction [11]).** *A deduction  $\pi_N = [C_1, \dots, C_n]$  of a clause  $C_n$  from some clause set  $N$  is a finite sequence of clauses such that for each  $C_i$  the following holds:*

- 1.1  $C_i$  is a renamed, variable-fresh version of a clause in  $N$ , or
- 1.2 there is a clause  $C_j \in \pi_N$ ,  $j < i$  s.t.  $C_i$  is the result of a Factoring inference from  $C_j$ , or
- 1.3 there are clauses  $C_j, C_k \in \pi_N$ ,  $j < k < i$  s.t.  $C_i$  is the result of a Resolution inference from  $C_j$  and  $C_k$ ,

and for each  $C_i \in \pi_N$ ,  $i < n$ :

- 2.1 there exists exactly one factor  $C_j$  of  $C_i$  with  $j > i$ , or  
 2.2 there exists exactly one  $C_j$  and  $C_k$  such that  $C_k$  is a resolvent of  $C_i$  and  $C_j$  and  $i, j < k$ .

We omit the subscript  $N$  in  $\pi_N$  if the context is clear.

A deduction  $\pi'$  of some clause  $C \in \pi$ , where  $\pi, \pi'$  are deductions from  $N$  is a subdeduction of  $\pi$  if  $\pi' \subseteq \pi$ , where the subset relation is overloaded for sequences. A deduction  $\pi_N = [C_1, \dots, C_{n-1}, \perp]$  is called a *refutation*. While the conditions 3.1.1, 3.1.2, and 3.1.3 are sufficient to represent a resolution derivation, the conditions 3.2.1 and 3.2.2 force deductions to be minimal with respect to  $C_n$ .

Note that variable renamings are only applied to clauses from  $N$  such that all clauses from  $N$  that are introduced in the deduction are variable disjoint. Also recall that our notion of a deduction implies a tree structure. Both assumptions together admit the existence of overall grounding substitutions for a deduction.

**Definition 4 (Overall Substitution of a Deduction [11]).** *Given a deduction  $\pi$  of a clause  $C_n$  the overall substitution  $\tau_{\pi,i}$  of  $C_i \in \pi$  is recursively defined by*

- 1 if  $C_i$  is a factor of  $C_j$  with  $j < i$  and mgu  $\sigma$ , then  $\tau_{\pi,i} = \tau_{\pi,j} \circ \sigma$ ,
- 2 if  $C_i$  is a resolvent of  $C_j$  and  $C_k$  with  $j < k < i$  and mgu  $\sigma$ , then  $\tau_{\pi,i} = (\tau_{\pi,j} \circ \tau_{\pi,k}) \circ \sigma$ ,
- 3 if  $C_i$  is an initial clause, then  $\tau_{\pi,i} = \emptyset$ ,

and the overall substitution of the deduction is  $\tau_\pi = \tau_{\pi,n}$ . We omit the subscript  $\pi$  if the context is clear.

Overall substitutions are well-defined because clauses introduced from  $N$  into the deduction are variable disjoint and each clause is used exactly once in the deduction. A grounding of an overall substitution  $\tau$  of some deduction  $\pi$  is a substitution  $\tau\delta$  such that  $\text{codom}(\tau\delta)$  only contains ground terms and  $\text{dom}(\delta)$  is exactly the variables from  $\text{codom}(\tau)$ .

**Definition 5 (SOS Deduction [11]).** *A deduction  $\pi_{N \cup S} = [C_1, \dots, C_n]$  is called an SOS deduction with SOS  $S$ , if the derivation  $(N, S_0) \Rightarrow_{RES}^* (N, S_m)$  is an SOS derivation where  $C'_1, \dots, C'_m$  is the subsequence from  $[C_1, \dots, C_n]$  with input clauses removed,  $S_0 = S$ , and  $S_{i+1} = S_i \cup C'_{i+1}$ .*

Oftentimes, it is of particular interest to identify the set of clauses that is minimally unsatisfiable, i.e., removing a clause would make it satisfiable. The earliest mention of such a notion is in [26] where it is introduced via a decision problem. Minimally unsatisfiable sets (MUS) have also gained a lot of attention in practice.

**Definition 6 (Minimal Unsatisfiable Subset (MUS) [20]).** *Given an unsatisfiable set of clauses  $N$ , the subset  $N' \subseteq N$  is a minimally unsatisfiable subset (MUS) of  $N$  if any strict subset of  $N'$  is satisfiable.*

In our previous work, we defined a notion of relevance based on how clauses may contribute to unsatisfiability by means of refutations.

**Definition 7 (Syntactic Relevance [11]).** *Given an unsatisfiable set of clauses  $N$ , a clause  $C \in N$  is syntactically relevant if for all refutations  $\pi$  of  $N$  it holds that  $C \in \pi$ . A clause  $C \in N$  is syntactically semi-relevant if there exists a refutation  $\pi$  of  $N$  in which  $C \in \pi$ . A clause  $C \in N$  is syntactically irrelevant if there is no refutation  $\pi$  of  $N$  in which  $C \in \pi$ .*

Syntactic relevance can be identified by using the resolution calculus. A clause  $C \in N$  is syntactically semi-relevant if and only if there exists an SOS refutation from  $\text{SOS } \{C\}$  and  $N \setminus \{C\}$ .

**Theorem 8 (Syntactic Relevance [11]).** *Given an unsatisfiable set of clauses  $N$ , the clause  $C \in N$  is*

1. *syntactically relevant if and only if  $N \setminus \{C\}$  is satisfiable,*
2. *syntactically semi-relevant if and only if  $(N \setminus \{C\}, \{C\}) \Rightarrow_{RES}^* (N \setminus \{C\}, S \cup \{\perp\})$ .*

An open problem from [11] is the question of a semantic counterpart to syntactic semi-relevance. Without any further properties of the clause set  $N$ , the notion of semi-relevance can lead to unintuitive results. For example, a tautology could be semi-relevant. Given a refutation showing semi-relevance of some clause  $C$ , where, in the refutation, some unary predicate  $P$  occurs, the refutation can be immediately extended using the tautology  $P(x) \vee \neg P(x)$ . We may additionally stumble upon a problem in the case where our set of clauses contains a subsumed clause. For example, if both clauses  $Q(a)$  and  $Q(x)$  exist in a clause set, they may be both semi-relevant, although from an intuition point of view one may only want to consider  $Q(x)$  to be semi-relevant, or even relevant. On the other hand, in some cases, redundant clauses are welcome as semi-relevant clauses.

*Example 9 (Redundant Clauses).* Given a set of clauses

$$N = \{Q(x), \quad Q(a), \quad \neg Q(a) \vee P(b), \quad \neg P(b), \quad P(x) \vee \neg P(x)\},$$

all clauses are syntactically semi-relevant while  $\neg Q(a) \vee P(b)$  and  $\neg P(b)$  are syntactically relevant. However, if we disregard the redundant clauses  $Q(a)$  and  $P(x) \vee \neg P(x)$ , then the clause  $Q(x)$  becomes a relevant clause. Therefore, for our semantic notion of relevance we will only consider clause sets without clauses implied by other, different clauses from the clause set.

### 3 Semantic Relevance

Except for the trivially false clause  $\perp$ , the simplest form of a contradiction is two unit clauses  $K$  and  $L$  such that  $K$  and  $\text{comp}(L)$  are unifiable. They will be called *conflict literals*, below. Then the idea for our semantic definition of

semi-relevance is to consider clauses that contribute to the number of conflict literals of a clause set. Furthermore, we will show that in any MUS every literal is a conflict literal.

While conflict literals could straightforwardly be defined in propositional logic having the above idea in mind, in first-order logic we have always to relate properties of literals, clauses to their respective ground instances. This is simply due to the fact that unsatisfiability of a first-order clause set is given by unsatisfiability of a finite set of ground instances from this set. Eventually, we will show that for independent clause sets a clause is semi-relevant, if it contributes to the number of conflict literals.

**Definition 10 (Conflict Literal).** *Given a set of clauses  $N$  over some signature  $\Sigma$ , a ground literal  $L$  is a conflict literal in a clause set  $N$  if there are two satisfiable clause sets  $N_1, N_2$  such that*

1. *the clauses in  $N_1, N_2$  are instances of clauses from  $N$  and*
2.  *$N_1 \models L$  and  $N_2 \models \text{comp}(L)$ .*

$\text{conflict}(N)$  denotes the set of conflict literals in  $N$ .

Our notion of a conflict literal generalizes the respective notion in [12] defined for propositional logic.

*Example 11 (Conflict Literal).* Given an unsatisfiable set of clauses over the signature  $\Sigma = (\{a, b, c, d, f\}, \{P\})$ :

$$N = \{\neg P(f(a, x)) \vee \neg P(f(c, y)), P(f(x, d)) \vee P(f(y, b))\}$$

Consider the following satisfiable sets of instances from  $N$

$$N_1 = \{\neg P(f(a, d)) \vee \neg P(f(c, y)), P(f(x, d)) \vee P(f(a, b))\}$$

$$N_2 = \{\neg P(f(a, b)) \vee \neg P(f(c, y)), P(f(x, d)) \vee P(f(c, b))\}$$

$P(f(a, b))$  is a conflict literal because  $N_1 \models P(f(a, b))$  and  $N_2 \models \neg P(f(a, b))$ .

We can show that  $N_1 \models P(f(a, b))$  because the resolution calculus is sound. Resolving both literals of  $\neg P(f(a, d)) \vee \neg P(f(c, y))$  with the first literal of the clause  $P(f(x, d)) \vee P(f(a, b))$  results in the clause  $P(f(a, b)) \vee P(f(a, b))$  which can be factorized to  $P(f(a, b))$ . Moreover,  $N_1$  is satisfiable: An interpretation  $\mathcal{I}$  with  $\mathcal{I}(P(f(a, b))) = 1$  and  $\mathcal{I}(P(t)) = 0$  for all terms  $t \neq f(a, b)$  satisfies  $N_1$  and  $P(f(a, b))$ .  $N_2 \models \neg P(f(a, b))$  can also be shown in the same manner.

*Example 12 (Conflict Literal).* Given

$$\begin{aligned} N = \{ & \neg R(z), R(c) \vee P(a, y), \\ & Q(a), \neg Q(x) \vee P(x, b), \\ & \neg P(a, b)\} \end{aligned}$$

its conflict literals are

$$\begin{aligned} \text{conflict}(N) = \{ & P(a, b), \neg P(a, b), \\ & R(c), \neg R(c), \\ & Q(a), \neg Q(a) \} \end{aligned}$$

In addition to a refutation, the existence of a conflict literal is another way to characterize unsatisfiability of a clause set. Obviously, conflict literals always come in pairs.

**Lemma 13 (Minimal Unsatisfiable Ground Clause Sets and Conflict Literals).** *If  $N$  is a minimally unsatisfiable set of ground clauses (MUS) then any literal occurring in  $N$  is a conflict literal.*

*Proof* Take any ground atom  $A$  such that  $A$  occurs in  $N$ .  $N$  can be split into three disjoint clause sets:

$$\begin{aligned} N_\emptyset &= \{C \in N \mid A \notin C \text{ and } \neg A \notin C\} \\ N_A &= \{C \in N \mid A \in C\} \\ N_{\neg A} &= \{C \in N \mid \neg A \in C\} \end{aligned}$$

Since  $N$  is minimal,  $N_A$  and  $N_{\neg A}$  are nonempty, because otherwise  $A$  is a pure literal and its corresponding clauses can be removed from  $N$  preserving unsatisfiability. Obviously  $N_\emptyset \cup N_A$  must be satisfiable, for otherwise the initial choice of  $N$  was not minimal. However,  $N_\emptyset \cup N'_A$ , where  $N'_A$  results from all  $N_A$  by deleting all  $A$  literals from the clauses of  $N_A$ , must be unsatisfiable, for otherwise we can construct a satisfying interpretation for  $N$ . Thus, every model of  $N_\emptyset \cup N_A$  must also be a model of  $A$ :  $N_\emptyset \cup N_A \models A$ . Using the same argument,  $N_\emptyset \cup N_{\neg A}$  is satisfiable and  $N_\emptyset \cup N_{\neg A} \models \neg A$ . Therefore,  $A$  is a conflict literal.  $\square$

**Lemma 14 (Conflict Literals and Unsatisfiability).** *Given a set of clauses  $N$ ,  $\text{conflict}(N) \neq \emptyset$  if and only if  $N$  is unsatisfiable.*

*Proof* “ $\Rightarrow$ ” Let  $L \in \text{conflict}(N)$ . By definition, there are two satisfiable subsets of instances  $N_1, N_2$  from  $N$  such that  $N_1 \models L$  and  $N_2 \models \text{comp}(L)$ . Towards contradiction, suppose  $N$  is satisfiable. Then, there exists an interpretation  $\mathcal{I}$  with  $\mathcal{I} \models N$  and therefore it holds that  $\mathcal{I} \models N_1$  and  $\mathcal{I} \models N_2$ . Furthermore, by definition of a conflict literal,  $\mathcal{I} \models L$  and  $\mathcal{I} \models \text{comp}(L)$ , a contradiction.

“ $\Leftarrow$ ” Given an unsatisfiable clause set  $N$ , we show that there is a conflict literal in  $N$ . Since  $N$  is unsatisfiable, by compactness of first-order logic there is a minimal set of ground instances  $N'$  from  $N$  that is also unsatisfiable. The rest follows from Lemma 13.  $\square$

Intuitively, a clause that is implied by other clauses is redundant and can be removed from the set of clauses. However, then applying a calculus generating new clauses, this intuitive notion of redundancy may destroy completeness [2, 23]. Still, the detection and elimination of redundant clauses, compatible or incompatible with completeness, is an important concept to the efficiency of automatic



reasoning, e.g., in propositional logic [3, 18]. It is also apparently important when we try to define a semantic notion of relevance. For example, a syntactically relevant clause would step down to be syntactically semi-relevant if it is duplicated. So, in order to have a semantically robust notion of relevance in first-order logic, we need to use a strong notion of (in)dependency.

**Definition 15 (Dependency).** *A clause  $C$  is dependent in  $N$  if there exists a satisfiable set of instances  $N'$  from  $N \setminus \{C\}$  such that  $N' \models C\sigma$  for some  $\sigma$ . If  $C$  is not dependent in  $N$  it is independent in  $N$ . A clause set  $N$  is independent if it does not contain any dependent clauses.*

A subsumed clause is obviously a dependent clause. However, there could also be non-subsumed clauses that are dependent. For example, in the set of clauses

$$N = \{P(a, y), P(x, b), \neg P(a, b)\}$$

$P(x, b)$  is dependent because  $P(a, b)$  is an instance of  $P(x, b)$  and it is entailed by  $P(a, y)$ . Now, we are ready to define the semantic notion of relevance based on conflict literals and dependency.

In some way, our notion of independence of clause sets is a strong assumption because there might be non-redundant clauses that are considered dependent. While this holds by design in some scenarios (e.g. the mentioned car scenario) in others it is violated by design. In addition, one question that may arise is how to acquire an independent clause set out of a dependent one. For example, in a scenario where some theory is developed out of some independent axioms. Then of course proven lemmas, theorems are dependent with respect to the axioms. In this case one could trace out of the proofs the dependency relations between the intermediate lemmas, theorems and the axioms and this way calculate independent clause sets with respect to some proven conjecture. This would then lead again to independent (sub) clause sets with respect to the proven conjecture where our results are applicable.

**Definition 16 (Semantic Relevance).** *Given an unsatisfiable set of independent clauses  $N$ , a clause  $C \in N$  is*

1. relevant, if  $\text{conflict}(N \setminus \{C\}) = \emptyset$
2. semi-relevant, if  $\text{conflict}(N \setminus \{C\}) \subsetneq \text{conflict}(N)$
3. irrelevant, if  $\text{conflict}(N \setminus \{C\}) = \text{conflict}(N)$

*Example 17 (Dependent Clauses in Propositional Logic).*

$$\begin{aligned} N = \{ & P, \neg P, \\ & \neg P \vee Q, \neg R \vee P, \\ & \neg Q \vee R \} \end{aligned}$$

The existence of dependent clauses  $\neg P \vee Q$  and  $\neg R \vee P$  causes an independent clause  $\neg Q \vee R$  to be a semi-relevant clause. However,  $\neg Q \vee R$  is not inside the only MUS  $\{P, \neg P\}$ .

Very often, concepts from propositional logic can be generalized to first-order logic. However, in the context of relevance this is not the case. Our notion of (semi-)relevance can also be characterized by MUSes in propositional logic, but not in first-order logic without considering instances of clauses.

**Lemma 18 (Propositional Clause Sets and Relevance).** *Given an independent unsatisfiable set of propositional clauses  $N$ , the relevant clauses coincide with the intersection of all MUSes and the semi-relevant clauses coincide with the union of all MUSes.*

*Proof* For the case of relevance: Given  $C \in N$ ,  $C$  is relevant if and only if  $\text{conflict}(N \setminus \{C\}) = \emptyset$  if and only if  $N \setminus \{C\}$  is satisfiable by Lemma 14 if and only if  $C$  is contained in all MUSes  $N'$  of  $N$ .

For the case of semi-relevance: Given  $C \in N$ , we show  $C$  is semi-relevant if and only if  $C$  is in some MUS  $N' \subseteq N$ .

“ $\Rightarrow$ ”: Towards contradiction, suppose there is a semi-relevant clause  $C$  that is not in any MUS. By definition of semi-relevant clauses, there are satisfiable sets  $N_1$  and  $N_2$  and a propositional variable  $P$  such that  $N_1 \models P$ ,  $N_2 \models \neg P$  but the MUS  $M$  out of  $N_1 \cup N_2$  does not contain  $C$ . By Theorem 2 there exist deductions  $\pi_1$  and  $\pi_2$  of  $P$  and  $\neg P$  from  $N_1$  and  $N_2$ , respectively. Since a deduction is connected, some clauses in  $M$  and  $(N_1 \cup N_2) \setminus M$  must have some complementary propositional literals  $Q$  and  $\neg Q$ , respectively to be eventually resolved upon in either  $\pi_1$  or  $\pi_2$ . At least one of these deductions must contain this resolution step between a clause from  $M$  and one from  $(N_1 \cup N_2) \setminus M$ . Now by Lemma 13 the literals  $Q$  and  $\neg Q$  are conflict literals in  $M$ . Thus, there are satisfiable subsets from  $M$  which entail  $Q$  and  $\neg Q$ , respectively. Therefore, the clause containing  $Q$  or  $\neg Q$  in  $(N_1 \cup N_2) \setminus M$  is dependent contradicting the assumption that  $N$  does not contain dependent clauses.

“ $\Leftarrow$ ”: If  $C$  is in some MUS  $N' \subseteq N$ , then,  $N' \setminus \{C\}$  is satisfiable. So invoking Lemma 13 any literal  $L \in C$  is a conflict literal in  $N'$ . In addition,  $L$  is not a conflict literal in  $N \setminus \{C\}$  for otherwise  $C$  is dependent: Suppose  $L$  is a conflict literal in  $N \setminus \{C\}$  then, by definition, there is satisfiable subset from  $N \setminus \{C\}$  which entails  $L$ . However, since  $L \models C$ , it means  $C$  is dependent.  $\square$

The next example demonstrates that the notion of a MUS cannot be carried over straightforwardly to the level of clauses with variables to characterize semi-relevant clauses in first-order logic.

*Example 19 (First-Order Relevant Clauses).* Given a set of clauses

$$N = \{P(a, y), \neg P(a, d) \vee Q(b, d), \\ \neg P(x, c), \neg Q(b, d) \vee P(d, c), Q(z, e)\}$$

over  $\Sigma = (\{a, b, c, d, e\}, \{P, Q\})$ . The conflict literals are

$$\{(\neg)P(a, c), (\neg)Q(b, d), (\neg)P(d, c), (\neg)P(a, d)\}.$$

The clause  $P(a, y)$  is relevant. The literals entailed by some satisfiable instances  $N'$  from  $N$  such that  $P(a, y) \notin N'$  are  $\{\neg Q(b, d)\} \uplus \{\neg P(t, c), \neg Q(t, e) \mid t \in \{a, b, c, d, e\}\}$  and no two of them are complementary. Thus,  $\text{conflict}(N \setminus \{P(a, y)\}) = \emptyset$ . The clause  $\neg P(a, d) \vee Q(b, d)$  is semi-relevant:  $Q(b, d) \notin \text{conflict}(N \setminus \{\neg P(a, d) \vee Q(b, d)\})$ . The clause  $Q(z, e)$  is irrelevant.

With respect to a MUS, the clause  $\neg P(a, d) \vee Q(b, d)$  from Example 19 is irrelevant. The only MUS from  $N$  is  $\{P(a, y), \neg P(x, c)\}$  with grounding substitution  $\{x \mapsto a, y \mapsto c\}$ . However, in first-order logic we should not ignore the clauses  $\neg P(a, d) \vee Q(b, d)$ ,  $\neg Q(b, d) \vee P(d, c)$ , because together with the clauses  $P(a, y), \neg P(x, c)$  they result in a different grounding  $\{x \mapsto d, y \mapsto d\}$ . So, we argue that MUS-based (semi-)relevance on the original clause set is not sufficient to characterize the way clauses are used to derive a contradiction for full first-order logic. However, it does so if ground instances are considered.

**Lemma 20 (Relevance and MUSes on First-Order Clauses).** *Given an unsatisfiable set of independent first-order clauses  $N$ . Then a clause  $C$  is relevant in  $N$ , if all MUSes of unsatisfiable sets of ground instances from  $N$  contain a ground instance of  $C$ . The clause  $C$  is semi-relevant in  $N$ , if there exists a MUS of an unsatisfiable set of ground instances from  $N$  that contains a ground instance of  $C$ .*

*Proof* (Relevance) Since all ground instances from  $N$  contain a ground instance of  $C$ , then, if  $N \setminus \{C\}$  contains a ground MUS from  $N$  it means that some ground instance of  $C$  is entailed by  $N \setminus \{C\}$ . This violates our assumption that  $N$  contains no dependent clauses. Thus,  $N \setminus \{C\}$  contains no ground MUSes. This further means that  $N \setminus \{C\}$  is satisfiable by the compactness theorem of first-order logic. By Lemma 14 it therefore has no conflict literals and  $C$  is relevant. (Semi-Relevance) Take some ground MUS  $M$  containing some ground instance  $C'$  of  $C$ . Due to Lemma 13, any literal  $P \in C'$  is a conflict literal in  $M$  and consequently also in  $N$ . In addition,  $P$  is not a conflict literal in  $N \setminus \{C\}$  for otherwise  $C$  is dependent: Suppose  $P$  is a conflict literal in  $N \setminus \{C\}$ . Then, by definition, there is some satisfiable instances from  $N \setminus \{C\}$  which entails  $P$ . However, since  $P \models C'$ , it means  $C$  is dependent. In conclusion,  $P \in \text{conflict}(N) \setminus \text{conflict}(N \setminus \{C\})$  and thus  $C$  is semi-relevant.  $\square$

In Example 19, we could identify two ground MUSes:

$$\{P(a, c), \neg P(a, c)\}$$

and

$$\{P(a, d), \neg P(a, d) \vee Q(b, d), \neg P(d, c), \neg Q(b, d) \vee P(d, c)\}$$

Our notion of relevance is thus alternatively explainable using Lemma 20:  $P(a, y)$  is relevant because every MUS contains an instance of it ( $P(a, c)$  and  $P(a, d)$ ). The clause  $\neg P(a, d) \vee Q(b, d)$  is semi-relevant as it is immediately contained in the second MUS. The clause  $Q(z, e)$  is irrelevant since no MUS contains any instance of  $Q(z, e)$ . On the other hand, we may still encounter the case where a dependent

clause is actually categorized as syntactically semi-relevant. Therefore, by using the dependency notion while at the same time not restricting a refutation to only use MUS as the input set, we can show that (semi-)relevance actually coincides with the syntactic (semi-)relevance. So, the semi-decidability result also follows.

**Theorem 21 (Semantic versus Syntactic Relevance).** *Given an independent, unsatisfiable set of clauses  $N$  in first-order logic, then (semi-)relevant clauses coincide with syntactically (semi-)relevant clauses.*

*Proof* We show the following: if  $N$  contains no dependent clause,  $C$  is (semi-)relevant if and only if  $C$  is syntactically (semi-)relevant. The case for relevant clauses is a consequence of Lemma 14. Now, we show it for semi-relevant clauses. “ $\Rightarrow$ ” Let  $L$  be a ground literal with  $L \in \text{conflict}(N) \setminus \text{conflict}(N \setminus \{C\})$ . We can construct a refutation using  $C$ . There are two satisfiable subsets of instances  $N_1, N_2$  from  $N$  such that  $N_1 \models L$  and  $N_2 \models \text{comp}(L)$  where  $N_1 \cup N_2$  contains at least one instance of  $C$ , for otherwise  $L \notin \text{conflict}(N) \setminus \text{conflict}(N \setminus \{C\})$ . By the deductive completeness, Theorem 2, and the fact that  $L$  and  $\text{comp}(L)$  are ground literals, there are two variable disjoint deductions  $\pi_1$  and  $\pi_2$  of some literals  $K_1$  and  $K_2$  such that  $K_1\sigma = L$  and  $K_2\sigma = \text{comp}(L)$  for some grounding  $\sigma$ . Obviously, the two variable disjoint deductions can be combined to a refutation  $\pi_1.\pi_2.\perp$  containing  $C$ . Thus,  $C$  is syntactically semi-relevant in  $N$ .

“ $\Leftarrow$ ” Given an SOS refutation  $\pi$  using  $C$ , i.e., an SOS refutation  $\pi$  from  $N \setminus \{C\}$  with SOS  $\{C\}$  and overall grounding substitution  $\sigma$ , we show that  $C$  is semantically semi-relevant. Let  $N'$  be the variable renamed versions of clauses from  $N \setminus \{C\}$  used in the refutation and  $S'$  be the renamed copies of  $C$  used in the refutation. First, we show that  $N'\sigma$  is satisfiable. Towards contradiction, suppose  $N'\sigma$  is unsatisfiable and let  $M\sigma \subseteq N'\sigma$  be its MUS. Since  $\pi$  is connected, some clauses in  $M\sigma$  and  $S'\sigma \cup (N'\sigma \setminus M\sigma)$  contains literals  $L$  and  $\text{comp}(L)$  respectively. By Lemma 13,  $L$  and  $\text{comp}(L)$  are also conflict literals in  $M\sigma$ . So, by Definition 15, the clause containing  $\text{comp}(L)$  in  $S'\sigma \cup (N'\sigma \setminus M\sigma)$  is dependent violating our initial assumption.

Now, since  $N'\sigma$  is satisfiable, there is a ground MUS from  $(N' \cup S')\sigma$  containing some  $C'\sigma \in S\sigma$ . Due to Lemma 13, any  $L \in C'\sigma$  is a conflict literal in  $N'$  (and consequently also in  $N$ ). In addition,  $L$  is not a conflict literal in  $N \setminus \{C\}$  for otherwise  $C$  is dependent: Suppose  $L$  is a conflict literal in  $N \setminus \{C\}$ . Then, by definition, there is some satisfiable instances from  $N \setminus \{C\}$  which entails  $L$ . However, since  $L \models C'\sigma$ , it means  $C$  is dependent. In conclusion,  $L \in \text{conflict}(N) \setminus \text{conflict}(N \setminus \{C\})$  and thus  $C$  is semi-relevant.  $\square$

When we have a ground MUS, identification of conflict literals is obvious because all of the literals in it are. However, testing if a literal  $L$  is a conflict literal is not trivial, in general. One can try enumerating all MUSes and check if  $L$  is contained in some. This definitely works for propositional logic despite being computationally expensive. In first-order logic, this is problematic because there could potentially be an infinite number of MUSes and determining a MUS is not even semi-decidable, in general. The following lemma provides a semi-decidable test using the SOS strategy.

**Lemma 22** *Given a ground literal  $L$  and an unsatisfiable set of clauses  $N$  with no dependent clauses,  $L$  is a conflict literal if and only if there is an SOS refutation from  $(N, \{L \vee \text{comp}(L)\})$ .*

*Proof “ $\Rightarrow$ ”* By the deductive completeness, Theorem 2, and the fact that  $L$  and  $\text{comp}(L)$  are ground literals, there are two variable disjoint deductions  $\pi_1$  and  $\pi_2$  of some literals  $K_1$  and  $K_2$  such that  $K_1\sigma = L$  and  $K_2\sigma = \text{comp}(L)$  for some grounding  $\sigma$ . Obviously, the two variable disjoint deductions can be combined to a refutation  $\pi_1.\pi_2.\perp$ . We can then construct a refutation  $\pi_1.\pi_2.(L \vee \neg L).\text{comp}(L).\perp$  where  $K_2$  is resolved with  $L \vee \text{comp}(L)$  to get  $\text{comp}(L)$  which will be resolved with  $K_1$  from  $\pi_1$  to get  $\perp$ . By Theorem 7, it means there is an SOS refutation from  $(N, \{L \vee \neg L\})$

*“ $\Leftarrow$ ”* Given an SOS refutation  $\pi$  using  $\{L \vee \text{comp}(L)\}$ , i.e., an SOS refutation  $\pi$  from  $N \setminus \{\{L \vee \text{comp}(L)\}\}$  with SOS  $\{\{L \vee \text{comp}(L)\}\}$ , Let  $N'$  be the variable renamed versions of clauses from  $N$  and overall grounding substitution  $\sigma$ .  $N'\sigma$  is a MUS for otherwise there is a dependent clause: Suppose  $N'\sigma \setminus M$  is an MUS where  $M$  is non-empty. Since  $\pi$  is connected, some clause  $D'$  in  $M$  must be resolved with some  $D \in N'\sigma$  upon some literal  $K$ . Thus, by Lemma 13,  $K$  and  $\text{comp}(K)$  are also conflict literals in  $N'\sigma \setminus M$ . So, by Definition 15, the clause subsuming  $D'$  in  $N$  is dependent violating our initial assumption. Finally, because  $L$  occurs in  $N'\sigma$  and  $N'\sigma$  is an MUS, by Lemma 13,  $L$  is a conflict literal.  $\square$

## 4 Conclusion

The main results of this paper are: (i) a semantic notion of relevance based on the existence of conflict literals, Definition 10, and Definition 16, (ii) its relationship to syntactic relevance, namely, both notions coincide for independent clause sets, Theorem 21, and (iii) the relationship of semantic relevance to minimal unsatisfiable sets, MUSes, both for propositional logic, Lemma 18, and first-order logic, Lemma 20.

The semantic relevance notion sheds some further light on the way clauses may contribute to a refutation beyond what can be offered by the notion of MUSes. While the syntactic notion of semi-relevance also considers redundant clauses such as tautologies to be semi-relevant, the semantic notion rules out redundant clauses. Here, the notions only coincide for independent clause sets. Still, the syntactic notion is “easier” to test and there are applications where clause sets do not contain implied clauses by construction. Hence, the syntactic-relevance coincides with semantic relevance. For example, first-order toolbox formalizations have this property because every tool is formalized by its own distinct predicate. Still a goal, refutation, can be reached by the use of different tools. The classic example is the toolbox for car/truck/tractor building [8, 31].

**Acknowledgments.** This work was partly funded by DFG grant 389792660 as part of TRR 248. We thank Christopher Lynch and David Plaisted for a number of discussions on semantic relevance. We thank the anonymous reviewers for their constructive and detailed comments.

## References

1. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. *J. Log. Comput.* **20**(1), 5–34 (2010)
2. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. I, chap. 2, pp. 19–99. Elsevier, Amsterdam (2001)
3. Boufkhad, Y., Roussel, O.: Redundancy in random SAT formulas. In: Kautz, H.A., Porter, B.W. (eds.) *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 30 July - 3 August, 2000, Austin, Texas, USA, pp. 273–278. AAAI Press/The MIT Press (2000)
4. Bourgaux, C., Ozaki, A., Peñaloza, R., Predoiu, L.: Provenance for the description logic ELHr. In: Bessière, C. (ed.) *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 1862–1869. [ijcai.org](http://ijcai.org) (2020)
5. Cimatti, A., Griggio, A., Sebastiani, R.: A simple and flexible way of computing small unsatisfiable cores in SAT modulo theories. In: Marques-Silva, J., Sakallah, K.A. (eds.) *SAT 2007. LNCS*, vol. 4501, pp. 334–339. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-72788-0\\_32](https://doi.org/10.1007/978-3-540-72788-0_32)
6. Cimatti, A., Griggio, A., Sebastiani, R.: Computing small unsatisfiable cores in satisfiability modulo theories. *J. Artif. Intell. Res.* **40**, 701–728 (2011)
7. Eiter, T., Gottlob, G.: The complexity of logic-based abduction. *J. ACM* **42**(1), 3–42 (1995)
8. Fetzer, C., Weidenbach, C., Wischnewski, P.: Compliance, functional safety and fault detection by formal methods. In: Margaria, T., Steffen, B. (eds.) *ISoLA 2016. LNCS*, vol. 9953, pp. 626–632. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-47169-3\\_48](https://doi.org/10.1007/978-3-319-47169-3_48)
9. Guthmann, O., Strichman, O., Trostanetski, A.: Minimal unsatisfiable core extraction for SMT. In: Piskac, R., Talupur, M. (eds.) *2016 Formal Methods in Computer-Aided Design, FMCAD 2016*, Mountain View, CA, USA, October 3–6, 2016. pp. 57–64. IEEE (2016)
10. Haifani, F., Koopmann, P., Tournet, S., Weidenbach, C.: On a notion of relevance. In: Borgwardt, S., Meyer, T. (eds.) *Proceedings of the 33rd International Workshop on Description Logics (DL 2020)* Co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], 12th to 14th September 2020. *CEUR Workshop Proceedings*, vol. 2663. CEUR-WS.org (2020)
11. Haifani, F., Tournet, S., Weidenbach, C.: Generalized completeness for SOS resolution and its application to a new notion of relevance. In: Platzer, A., Sutcliffe, G. (eds.) *CADE 2021. LNCS (LNAI)*, vol. 12699, pp. 327–343. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_19](https://doi.org/10.1007/978-3-030-79876-5_19)
12. Jabbour, S., Ma, Y., Raddaoui, B., Sais, L.: Quantifying conflicts in propositional logic through prime implicates. *Int. J. Approx. Reason.* **89**, 27–40 (2017)
13. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Aberer, K. (ed.) *ASWC/ISWC -2007. LNCS*, vol. 4825, pp. 267–280. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-76298-0\\_20](https://doi.org/10.1007/978-3-540-76298-0_20)
14. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) *Handbook of Satisfiability*, Fron-

- tiers in Artificial Intelligence and Applications, vol. 185, pp. 339–401. IOS Press, Amsterdam (2009)
15. Kullmann, O.: Investigations on autark assignments. *Discret. Appl. Math.* **107**(1–3), 99–137 (2000)
  16. Kullmann, O., Lynce, I., Marques-Silva, J.: Categorisation of clauses in conjunctive normal forms: minimally unsatisfiable sub-clause-sets and the lean kernel. In: Biere, A., Gomes, C.P. (eds.) *SAT 2006*. LNCS, vol. 4121, pp. 22–35. Springer, Heidelberg (2006). [https://doi.org/10.1007/11814948\\_4](https://doi.org/10.1007/11814948_4)
  17. Lee, C.T.: A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms. Ph.D. thesis, University of Berkeley, California, Department of Electrical Engineering (1967)
  18. Liberatore, P.: Redundancy in logic I: CNF propositional formulae. *Artif. Intell.* **163**(2), 203–232 (2005)
  19. Liu, S., Luo, J.: FMUS2: an efficient algorithm to compute minimal unsatisfiable subsets. In: Fleuriet, J., Wang, D., Calmet, J. (eds.) *AISC 2018*. LNCS (LNAI), vol. 11110, pp. 104–118. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-99957-9\\_7](https://doi.org/10.1007/978-3-319-99957-9_7)
  20. Marques-Silva, J., Mencía, C.: Reasoning about inconsistent formulas. In: Bessiere, C. (ed.) *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 4899–4906. [ijcai.org](http://ijcai.org) (2020)
  21. Mencía, C., Kullmann, O., Ignatiev, A., Marques-Silva, J.: On computing the union of MUSes. In: Janota, M., Lynce, I. (eds.) *SAT 2019*. LNCS, vol. 11628, pp. 211–221. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-24258-9\\_15](https://doi.org/10.1007/978-3-030-24258-9_15)
  22. Nienhuys-Cheng, S.-H., de Wolf, R.: The equivalence of the subsumption theorem and the refutation-completeness for unconstrained resolution. In: Kanchanasut, K., Lévy, J.-J. (eds.) *ACSC 1995*. LNCS, vol. 1023, pp. 269–285. Springer, Heidelberg (1995). [https://doi.org/10.1007/3-540-60688-2\\_50](https://doi.org/10.1007/3-540-60688-2_50)
  23. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. I, chap. 7, pp. 371–443. Elsevier, Amsterdam (2001)
  24. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. 1, chap. 6, pp. 335–367. Elsevier, Amsterdam (2001)
  25. Papadimitriou, C.H., Wolfe, D.: The complexity of facets resolved. *J. Comput. Syst. Sci.* **37**(1), 2–13 (1988)
  26. Papadimitriou, C.H., Yannakakis, M.: The complexity of facets (and some facets of complexity). *J. Comput. Syst. Sci.* **28**(2), 244–259 (1984)
  27. Peñaloza, R., Mencía, C., Ignatiev, A., Marques-Silva, J.: Lean kernels in description logics. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) *ESWC 2017*. LNCS, vol. 10249, pp. 518–533. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58068-5\\_32](https://doi.org/10.1007/978-3-319-58068-5_32)
  28. Robinson, J.A.: A machine-oriented logic based on the resolution principle. *J. ACM* **12**(1), 23–41 (1965)
  29. Robinson, J.A., Voronkov, A. (eds.): *Handbook of Automated Reasoning* (in 2 volumes). Elsevier and MIT Press, Cambridge (2001)
  30. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI-03*, Acapulco, Mexico, 9–15 August 2003, pp. 355–362. Morgan Kaufmann (2003)

31. Sinz, C., Kaiser, A., Küchlin, W.: Formal methods for the validation of automotive product configuration data. *Artif. Intell. Eng. Des. Anal. Manuf.* **17**(1), 75–97 (2003)
32. Walter, R., Felfernig, A., Küchlin, W.: Constraint-based and SAT-based diagnosis of automotive configuration problems. *J. Intell. Inf. Syst.* **49**(1), 87–118 (2016). <https://doi.org/10.1007/s10844-016-0422-7>
33. Wos, L., Robinson, G., Carson, D.: Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM* **12**(4), 536–541 (1965)
34. Xie, H., Luo, J.: An algorithm to compute minimal unsatisfiable subsets for a decidable fragment of first-order formulas. In: 28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, 6–8 November 2016, pp. 444–451. IEEE Computer Society (2016)
35. Zhang, J., Xu, W., Zhang, J., Shen, S., Pang, Z., Li, T., Xia, J., Li, S.: Finding first-order minimal unsatisfiable cores with a heuristic depth-first-search algorithm. In: Yin, H., Wang, W., Rayward-Smith, V. (eds.) IDEAL 2011. LNCS, vol. 6936, pp. 178–185. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-23878-9\\_22](https://doi.org/10.1007/978-3-642-23878-9_22)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.



The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.







# SCL(EQ): SCL for First-Order Logic with Equality

Hendrik Leidinger<sup>1,2</sup>(✉)  and Christoph Weidenbach<sup>1</sup> 

<sup>1</sup> Max-Planck Institute for Informatics, Saarbrücken, Germany  
{hleiding,weidenbach}@mpi-inf.mpg.de

<sup>2</sup> Graduate School of Computer Science, Saarbrücken, Germany

**Abstract.** We propose a new calculus SCL(EQ) for first-order logic with equality that only learns non-redundant clauses. Following the idea of CDCL (Conflict Driven Clause Learning) and SCL (Clause Learning from Simple Models) a ground literal model assumption is used to guide inferences that are then guaranteed to be non-redundant. Redundancy is defined with respect to a dynamically changing ordering derived from the ground literal model assumption. We prove SCL(EQ) sound and complete and provide examples where our calculus improves on superposition.

**Keywords:** First-order logic with equality · Term rewriting · Model-based reasoning

## 1 Introduction

There has been extensive research on sound and complete calculi for first-order logic with equality. The current prime calculus is superposition [2], where ordering restrictions guide paramodulation inferences and an abstract redundancy notion enables a number of clause simplification and deletion mechanisms, such as rewriting or subsumption. Still this “syntactic” form of superposition infers many redundant clauses. The completeness proof of superposition provides a “semantic” way of generating only non-redundant clauses, however, the underlying ground model assumption cannot be effectively computed in general [31]. It requires an ordered enumeration of infinitely many ground instances of the given clause set, in general. Our calculus overcomes this issue by providing an effective way of generating ground model assumptions that then guarantee non-redundant inferences on the original clauses with variables.

The underlying ordering is based on the order of ground literals in the model assumption, hence changes during a run of the calculus. It incorporates a standard rewrite ordering. For practical redundancy criteria this means that both rewriting and redundancy notions that are based on literal subset relations are permitted to dynamically simplify or eliminate clauses. Newly generated clauses are non-redundant, so redundancy tests are only needed backwards. Furthermore, the ordering is automatically generated by the structure of the clause set.

Instead of a fixed ordering as done in the superposition case, the calculus finds and changes an ordering according to the currently easiest way to make progress, analogous to CDCL (Conflict Driven Clause Learning) [11, 21, 25, 29, 34].

Typical for CDCL and SCL (Clause Learning from Simple Models) [1, 14, 18] approaches to reasoning, the development of a model assumption is done by decisions and propagations. A decision guesses a ground literal to be true whereas a propagation concludes the truth of a ground literal through an otherwise false clause. While propagations in CDCL and propositional logic are restricted to the finite number of propositional variables, in first-order logic there can already be infinite propagation sequences [18]. In order to overcome this issue, model assumptions in SCL(EQ) are at any point in time restricted to a finite number of ground literals, hence to a finite number of ground instances of the clause set at hand. Therefore, without increasing the number of considered ground literals, the calculus either finds a refutation or runs into a *stuck state* where the current model assumption satisfies the finite number of ground instances. In this case one can check whether the model assumption can be generalized to a model assumption of the overall clause set or the information of the stuck state can be used to appropriately increase the number of considered ground literals and continue search for a refutation. SCL(EQ) does not require exhaustive propagation, in general, it just forbids the decision of the complement of a literal that could otherwise be propagated.

For an example of SCL(EQ) inferring clauses, consider the three first-order clauses

$$\begin{aligned} C_1 &:= h(x) \approx g(x) \vee c \approx d & C_2 &:= f(x) \approx g(x) \vee a \approx b \\ C_3 &:= f(x) \not\approx h(x) \vee f(x) \not\approx g(x) \end{aligned}$$

with a Knuth-Bendix Ordering (KBO), unique weight 1, and precedence  $d \prec c \prec b \prec a \prec g \prec h \prec f$ . A Superposition Left [2] inference between  $C_2$  and  $C_3$  results in

$$C'_4 := h(x) \not\approx g(x) \vee f(x) \not\approx g(x) \vee a \approx b.$$

For SCL(EQ) we start by building a partial model assumption, called a *trail*, with two decisions

$$\Gamma := [h(a) \approx g(a)^{1:(h(x) \approx g(x) \vee h(x) \not\approx g(x)) \cdot \sigma}, f(a) \approx g(a)^{2:(f(x) \approx g(x) \vee f(x) \not\approx g(x)) \cdot \sigma}]$$

where  $\sigma := \{x \mapsto a\}$ . Decisions and propagations are always ground instances of literals from the first-order clauses, and are annotated with a level and a justification clause, in case of a decision a tautology. Now with respect to  $\Gamma$  clause  $C_3$  is false with grounding  $\sigma$ , and rule Conflict is applicable; see Sect. 3.1 for details on the inference rules. In general, clauses and justifications are considered variable disjoint, but for simplicity of the presentation of this example, we repeat variable names here as long as the same ground substitution is shared. The maximal literal in  $C_3\sigma$  is  $(f(x) \not\approx h(x))\sigma$  and a rewrite refutation using the ground equations from the trail results in the justification clause

$$(g(x) \not\approx g(x) \vee f(x) \not\approx g(x) \vee f(x) \not\approx g(x) \vee h(x) \not\approx g(x)) \cdot \sigma$$

where for the refutation justification clauses and all otherwise inferred clauses we use the grounding  $\sigma$  for guidance, but operate on the clauses with variables. The respective ground clause is smaller than  $(f(x) \not\approx h(x))\sigma$ , false with respect to  $\Gamma$  and becomes our new conflict clause by an application of our inference rule Explore-Refutation. It is simplified by our inference rules Equality-Resolution and Factorize, resulting in the finally learned clause

$$C_4 := h(x) \not\approx g(x) \vee f(x) \not\approx g(x)$$

which is then used to apply rule Backtrack to the trail. Observe that  $C_4$  is strictly stronger than  $C'_4$  the clause inferred by superposition and that  $C_4$  cannot be inferred by superposition. Thus SCL(EQ) can infer stronger clauses than superposition for this example.

*Related Work:* SCL(EQ) is based on ideas of SCL [1, 14, 18] but for the first time includes a native treatment of first-order equality reasoning. Similar to [14] propagations need not to be exhaustively applied, the trail is built out of decisions and propagations of ground literals annotated by first-order clauses, SCL(EQ) only learns non-redundant clauses, but for the first time conflicts resulting out of a decision have to be considered, due to the nature of the equality relation.

There have been suggested several approaches to lift the idea of an inference guiding model assumption from propositional to full first-order logic [6, 12, 13, 18]. They do not provide a native treatment of equality, e.g., via paramodulation or rewriting.

Baumgartner et al. describe multiple calculi that handle equality by using unit superposition style inference rules and are based on either hyper tableaux [5] or DPLL [15, 16]. Hyper tableaux fix a major problem of the well-known free variable tableaux, namely the fact that free variables within the tableau are rigid, i.e., substitutions have to be applied to all occurrences of a free variable within the entire tableau. Hyper tableaux with equality [7] in turn integrates unit superposition style inference rules into the hyper tableau calculus.

Another approach that is related to ours is the model evolution calculus with equality ( $\mathcal{ME}_E$ ) by Baumgartner et al. [8, 9] which lifts the DPLL calculus to first-order logic with equality. Similar to our approach,  $\mathcal{ME}_E$  creates a candidate model until a clause instance contradicts this model or all instances are satisfied by the model. The candidate model results from a so-called context, which consists of a finite set of non-ground rewrite literals. Roughly speaking, a context literal specifies the truth value of all its ground instances unless a more specific literal specifies the complement. Initially the model satisfies the identity relation over the set of all ground terms. Literals within a context may be universal or parametric, where universal literals guarantee all its ground instances to be true. If a clause contradicts the current model, it is repaired by a non-deterministic split which adds a parametric literal to the current model. If the added literal does not share any variables in the contradictory clause it is added as a universal literal.

Another approach by Baumgartner and Waldmann [10] combined the superposition calculus with the Model Evolution calculus with equality. In this cal-

culus the atoms of the clauses are labeled as “split atoms” or “superposition atoms”. The superposition part of the calculus then generates a model for the superposition atoms while the model evolution part generates a model for the split atoms. Conversely, this means that if all atoms are labeled as “split atom”, the calculus behaves similar to the model evolution calculus. If all atoms are labeled as “superposition atom”, it behaves like the superposition calculus.

Both the hyper tableaux calculus with equality and the model evolution calculus with equality allow only unit superposition applications, while SCL(EQ) inferences are guided paramodulation inferences on clauses of arbitrary length. The model evolution calculus with equality was revised and implemented in 2011 [8] and compares its performance with that of hyper tableaux. Model evolution performed significantly better, with more problems solved in all relevant TPTP [30] categories, than the implementation of the hyper tableaux calculus.

Plaisted et al. [27] present the Ordered Semantic Hyper-Linking (OSHL) calculus. OSHL is an instantiation based approach that repeatedly chooses ground instances of a non-ground input clause set such that the current model does not satisfy the current ground clause set. A further step repairs the current model such that it satisfies the ground clause set again. The algorithm terminates if the set of ground clauses contains the empty clause. OSHL supports rewriting and narrowing, but only with unit clauses. In order to handle non-unit clauses it makes use of other mechanisms such as Brand’s Transformation [3].

Inst-Gen [22] is an instantiation based calculus, that creates ground instances of the input first-order formulas which are forwarded to a SAT solver. If a ground instance is unsatisfiable, then the first-order set is as well. If not then the calculus creates more instances. The Inst-Gen-EQ calculus [23] creates instances by extracting instantiations of unit superposition refutations of selected literals of the first-order clause set. The ground abstraction is then extended by the extracted clauses and an SMT solver then checks the satisfiability of the resulting set of equational and non-equational ground literals.

In favor of examples and explanations we omit all proofs. They are available in an extended version published as a research report [24]. The rest of the paper is organized as follows. Section 2 provides basic formalisms underlying SCL(EQ). The rules of the calculus are presented in Sect. 3. Soundness and completeness results are provided in Sect. 4. We end with a discussion of obtained results and future work, Sect. 5. The main contribution of this paper is the SCL(EQ) calculus that only learns non-redundant clauses, permits subset based redundancy elimination and rewriting, and its soundness and completeness.

## 2 Preliminaries

We assume a standard first-order language with equality and signature  $\Sigma = (\Omega, \emptyset)$  where the only predicate symbol is equality  $\approx$ .  $N$  denotes a set of clauses,  $C, D$  denote clauses,  $L, K, H$  denote equational literals,  $A, B$  denote equational atoms,  $t, s$  terms from  $T(\Omega, \mathcal{X})$  for an infinite set of variables  $\mathcal{X}$ ,  $f, g, h$  function symbols from  $\Omega$ ,  $a, b, c$  constants from  $\Omega$  and  $x, y, z$  variables from  $\mathcal{X}$ . The function *comp* denotes the complement of a literal. We write  $s \not\approx t$  as a shortcut for

$\neg(s \approx t)$ . The literal  $s \# t$  may denote both  $s \approx t$  and  $s \not\approx t$ . The semantics of first-order logic and semantic entailment  $\models$  is defined as usual.

By  $\sigma, \tau, \delta$  we denote substitutions, which are total mappings from variables to terms. Let  $\sigma$  be a substitution, then its finite domain is defined as  $\text{dom}(\sigma) := \{x \mid x\sigma \neq x\}$  and its codomain is defined as  $\text{codom}(\sigma) = \{t \mid x\sigma = t, x \in \text{dom}(\sigma)\}$ . We extend their application to literals, clauses and sets of such objects in the usual way. A term, literal, clause or sets of these objects is ground if it does not contain any variable. A substitution  $\sigma$  is *ground* if  $\text{codom}(\sigma)$  is ground. A substitution  $\sigma$  is *grounding* for a term  $t$ , literal  $L$ , clause  $C$  if  $t\sigma$ ,  $L\sigma$ ,  $C\sigma$  is ground, respectively. By  $C \cdot \sigma$ ,  $L \cdot \sigma$  we denote a closure consisting of a clause  $C$ , literal  $L$  and a grounding substitution  $\sigma$ , respectively. The function  $\text{gnd}$  computes the set of all ground instances of a literal, clause, or clause set. The function  $\text{mgu}$  denotes the most general unifier of terms, atoms, literals, respectively. We assume that mgus do not introduce fresh variables and that they are idempotent.

The set of positions  $\text{pos}(L)$  of a literal (term  $\text{pos}(t)$ ) is inductively defined as usual. The notion  $L|_p$  denotes the subterm of a literal  $L$  ( $t|_p$  for term  $t$ ) at position  $p \in \text{pos}(L)$  ( $p \in \text{pos}(t)$ ). The replacement of a subterm of a literal  $L$  (term  $t$ ) at position  $p \in \text{pos}(L)$  ( $p \in \text{pos}(t)$ ) by a term  $s$  is denoted by  $L[s]_p$  ( $t[s]_p$ ). For example, the term  $f(a, g(x))$  has the positions  $\{\epsilon, 1, 2, 21\}$ ,  $f(a, g(x))|_{21} = x$  and  $f(a, g(x))[b]_2$  denotes the term  $f(a, b)$ .

Let  $R$  be a set of rewrite rules  $l \rightarrow r$ , called a *term rewrite system* (TRS). The rewrite relation  $\rightarrow_R \subseteq T(\Omega, \mathcal{X}) \times T(\Omega, \mathcal{X})$  is defined as usual by  $s \rightarrow_R t$  if there exists  $(l \rightarrow r) \in R$ ,  $p \in \text{pos}(s)$ , and a matcher  $\sigma$ , such that  $s|_p = l\sigma$  and  $t = s[r\sigma]_p$ . We write  $s = t \downarrow_R$  if  $s$  is the normal form of  $t$  in the rewrite relation  $\rightarrow_R$ . We write  $s \# t = (s' \# t') \downarrow_R$  if  $s$  is the normal form of  $s'$  and  $t$  is the normal form of  $t'$ . A rewrite relation is terminating if there is no infinite descending chain  $t_0 \rightarrow t_1 \rightarrow \dots$  and confluent if  $t \xrightarrow{*} s \rightarrow^* t'$  implies  $t \xrightarrow{*} t'$ . A rewrite relation is convergent if it is terminating and confluent. A rewrite order is a irreflexive and transitive rewrite relation. A TRS  $R$  is terminating, confluent, convergent, if the rewrite relation  $\rightarrow_R$  is terminating, confluent, convergent, respectively. A term  $t$  is called irreducible by a TRS  $R$  if no rule from  $R$  rewrites  $t$ . Otherwise it is called reducible. A literal, clause is irreducible if all of its terms are irreducible, and reducible otherwise. A substitution  $\sigma$  is called irreducible if any  $t \in \text{codom}(\sigma)$  is irreducible, and reducible otherwise.

Let  $\prec_T$  denote a well-founded rewrite ordering on terms which is total on ground terms and for all ground terms  $t$  there exist only finitely many ground terms  $s \prec_T t$ . We call  $\prec_T$  a *desired* term ordering. We extend  $\prec_T$  to equations by assigning the multiset  $\{s, t\}$  to positive equations  $s \approx t$  and  $\{s, s, t, t\}$  to inequations  $s \not\approx t$ . Furthermore, we identify  $\prec_T$  with its multiset extension comparing multisets of literals. For a (multi)set of terms  $\{t_1, \dots, t_n\}$  and a term  $t$ , we define  $\{t_1, \dots, t_n\} \prec_T t$  if  $\{t_1, \dots, t_n\} \prec_T \{t\}$ . For a (multi)set of Literals  $\{L_1, \dots, L_n\}$  and a term  $t$ , we define  $\{L_1, \dots, L_n\} \prec_T t$  if  $\{L_1, \dots, L_n\} \prec_T \{\{t\}\}$ . Given a ground term  $\beta$  then  $\text{gnd}_{\prec_T \beta}$  computes the set of all ground instances of a literal, clause, or clause set where the groundings are smaller than  $\beta$  according to

the ordering  $\prec_T$ . Given a set (sequence) of ground literals  $\Gamma$  let  $\text{conv}(\Gamma)$  be a convergent rewrite system out of the positive equations in  $\Gamma$  using  $\prec_T$ .

Let  $\prec$  be a well-founded, total, strict ordering on ground literals, which is lifted to clauses and clause sets by its respective multiset extension. We overload  $\prec$  for literals, clauses, clause sets if the meaning is clear from the context. The ordering is lifted to the non-ground case via instantiation: we define  $C \prec D$  if for all grounding substitutions  $\sigma$  it holds  $C\sigma \prec D\sigma$ . Then we define  $\preceq$  as the reflexive closure of  $\prec$  and  $N^{\preceq^C} := \{D \mid D \in N \text{ and } D \preceq C\}$  and use the standard superposition style notion of redundancy [2].

**Definition 1 (Clause Redundancy).** *A ground clause  $C$  is redundant with respect to a set  $N$  of ground clauses and an ordering  $\prec$  if  $N^{\preceq^C} \models C$ . A clause  $C$  is redundant with respect to a clause set  $N$  and an ordering  $\prec$  if for all  $C' \in \text{gnd}(C)$ ,  $C'$  is redundant with respect to  $\text{gnd}(N)$ .*

### 3 The SCL(EQ) Calculus

We start the introduction of the calculus by defining the ingredients of an SCL(EQ) state.

**Definition 2 (Trail).** *A trail  $\Gamma := [L_1^{i_1:C_1\cdot\sigma_1}, \dots, L_n^{i_n:C_n\cdot\sigma_n}]$  is a consistent sequence of ground equations and inequations where  $L_j$  is annotated by a level  $i_j$  with  $i_{j-1} \leq i_j$ , and a closure  $C_j\cdot\sigma_j$ . We omit the annotations if they are not needed in a certain context. A ground literal  $L$  is true in  $\Gamma$  if  $\Gamma \models L$ . A ground literal  $L$  is false in  $\Gamma$  if  $\Gamma \models \text{comp}(L)$ . A ground literal  $L$  is undefined in  $\Gamma$  if  $\Gamma \not\models L$  and  $\Gamma \not\models \text{comp}(L)$ . Otherwise it is defined. For each literal  $L_j$  in  $\Gamma$  it holds that  $L_j$  is undefined in  $[L_1, \dots, L_{j-1}]$  and irreducible by  $\text{conv}(\{L_1, \dots, L_{j-1}\})$ .*

The above definition of truth and undefinedness is extended to clauses in the obvious way. The notions of true, false, undefined can be parameterized by a ground term  $\beta$  by saying that  $L$  is  $\beta$ -undefined in a trail  $\Gamma$  if  $\beta \prec_T L$  or  $L$  is undefined. The notions of a  $\beta$ -true,  $\beta$ -false term are restrictions of the above notions to literals smaller  $\beta$ , respectively. All SCL(EQ) reasoning is layered with respect to a ground term  $\beta$ .

**Definition 3.** *Let  $\Gamma$  be a trail and  $L$  a ground literal such that  $L$  is defined in  $\Gamma$ . By  $\text{core}(\Gamma; L)$  we denote a minimal subsequence  $\Gamma' \subseteq \Gamma$  such that  $L$  is defined in  $\Gamma'$ . By  $\text{cores}(\Gamma; L)$  we denote the set of all cores.*

Note that  $\text{core}(\Gamma; L)$  is not necessarily unique. There can be multiple cores for a given trail  $\Gamma$  and ground literal  $L$ .

**Definition 4 (Trail Ordering).** *Let  $\Gamma := [L_1, \dots, L_n]$  be a trail. The (partial) trail ordering  $\prec_\Gamma$  is the sequence ordering given by  $\Gamma$ , i.e.,  $L_i \prec_\Gamma L_j$  if  $i < j$  for all  $1 \leq i, j \leq n$ .*

**Definition 5 (Defining Core and Defining Literal).** For a trail  $\Gamma$  and a sequence of literals  $\Delta \subseteq \Gamma$  we write  $\max_{\prec_\Gamma}(\Delta)$  for the largest literal in  $\Delta$  according to the trail ordering  $\prec_\Gamma$ . Let  $\Gamma$  be a trail and  $L$  a ground literal such that  $L$  is defined in  $\Gamma$ . Let  $\Delta \in \text{cores}(\Gamma; L)$  be a sequence of literals where  $\max_{\prec_\Gamma}(\Delta) \preceq_\Gamma \max_{\prec_\Gamma}(\Lambda)$  for all  $\Lambda \in \text{cores}(\Gamma; L)$ , then  $\max_\Gamma(L) := \max_{\prec_\Gamma}(\Delta)$  is called the defining literal and  $\Delta$  is called a defining core for  $L$  in  $\Gamma$ . If  $\text{cores}(\Gamma; L)$  contains only the empty core, then  $L$  has no defining literal and no defining core.

Note that there can be multiple defining cores but only one defining literal for any defined literal  $L$ . For example, consider a trail  $\Gamma := [f(a) \approx f(b)^{1:C_1 \cdot \sigma_1}, a \approx b^{2:C_2 \cdot \sigma_2}, b \approx c^{3:C_3 \cdot \sigma_3}]$  with an ordering  $\prec_T$  that orders the terms of the equations from left to right, and a literal  $g(f(a)) \approx g(f(c))$ . Then the defining cores are  $\Delta_1 := [a \approx b, b \approx c]$  and  $\Delta_2 := [f(a) \approx f(b), b \approx c]$ . The defining literal, however, is in both cases  $b \approx c$ . Defined literals that have no defining core and therefore no defining literal are literals that are trivially false or true. Consider, for example,  $g(f(a)) \approx g(f(a))$ . This literal is trivially true in  $\Gamma$ . Thus an empty subset of  $\Gamma$  is sufficient to show that  $g(f(a)) \approx g(f(a))$  is defined in  $\Gamma$ .

**Definition 6 (Literal Level).** Let  $\Gamma$  be a trail. A ground literal  $L \in \Gamma$  is of level  $i$  if  $L$  is annotated with  $i$  in  $\Gamma$ . A defined ground literal  $L \notin \Gamma$  is of level  $i$  if the defining literal of  $L$  is of level  $i$ . If  $L$  has no defining literal, then  $L$  is of level 0. A ground clause  $D$  is of level  $i$  if  $i$  is the maximum level of a literal in  $D$ .

The restriction to minimal subsequences for the defining literal and definition of a level eventually guarantee that learned clauses are smaller in the trail ordering. This enables completeness in combination with learning non-redundant clauses as shown later.

**Lemma 7.** Let  $\Gamma_1$  be a trail and  $K$  a defined literal that is of level  $i$  in  $\Gamma_1$ . Then  $K$  is of level  $i$  in a trail  $\Gamma := \Gamma_1, \Gamma_2$ .

**Definition 8.** Let  $\Gamma$  be a trail and  $L \in \Gamma$  a literal.  $L$  is called a decision literal if  $\Gamma = \Gamma_0, K^{i:C \cdot \tau}, L^{i+1:C' \cdot \tau'}, \Gamma_1$ . Otherwise  $L$  is called a propagated literal.

In our above example  $g(f(a)) \approx g(f(c))$  is of level 3 since the defining literal  $b \approx c$  is annotated with 3.  $a \approx b$  on the other hand is of level 2.

We define a well-founded total strict ordering which is induced by the trail and with which non-redundancy is proven in Sect. 4. Unlike SCL [14, 18] we use this ordering for the inference rules as well. In previous SCL calculi, conflict resolution automatically chooses the greatest literal and resolves with this literal. In SCL(EQ) this is generalized. Coming back to our running example above, suppose we have a conflict clause  $f(b) \not\approx f(c) \vee b \not\approx c$ . The defining literal for both inequations is  $b \approx c$ . So we could do paramodulation inferences with both literals. The following ordering makes this non-deterministic choice deterministic.

**Definition 9 (Trail Induced Ordering).** Let  $\Gamma := [L_1^{i_1:C_1 \cdot \sigma_1}, \dots, L_n^{i_n:C_n \cdot \sigma_n}]$  be a trail,  $\beta$  a ground term such that  $\{L_1, \dots, L_n\} \prec_T \beta$  and  $M_{i,j}$  all  $\beta$ -defined ground literals not contained in  $\Gamma \cup \text{comp}(\Gamma)$ : for a defining literal  $\max_\Gamma(M_{i,j}) = L_i$  and for two literals  $M_{i,j}, M_{i,k}$  we have  $j < k$  if  $M_{i,j} \prec_T M_{i,k}$ . The trail induces a total well-founded strict order  $\prec_{\Gamma^*}$  on  $\beta$ -defined ground literals  $M_{k,l}, M_{m,n}, L_i, L_j$  of level greater than zero, where

1.  $M_{i,j} \prec_{\Gamma^*} M_{k,l}$  if  $i < k$  or ( $i = k$  and  $j < l$ )
2.  $L_i \prec_{\Gamma^*} L_j$  if  $L_i \prec_\Gamma L_j$
3.  $\text{comp}(L_i) \prec_{\Gamma^*} L_j$  if  $L_i \prec_\Gamma L_j$
4.  $L_i \prec_{\Gamma^*} \text{comp}(L_j)$  if  $L_i \prec_\Gamma L_j$  or  $i = j$
5.  $\text{comp}(L_i) \prec_{\Gamma^*} \text{comp}(L_j)$  if  $L_i \prec_\Gamma L_j$
6.  $L_i \prec_{\Gamma^*} M_{k,l}, \text{comp}(L_i) \prec_{\Gamma^*} M_{k,l}$  if  $i \leq k$
7.  $M_{k,l} \prec_{\Gamma^*} L_i, M_{k,l} \prec_{\Gamma^*} \text{comp}(L_i)$  if  $k < i$

and for all  $\beta$ -defined literals  $L$  of level zero:

8.  $\prec_{\Gamma^*} := \prec_T$
9.  $L \prec_{\Gamma^*} K$  if  $K$  is of level greater than zero and  $K$  is  $\beta$ -defined

and can eventually be extended to  $\beta$ -undefined ground literals  $K, H$  by

10.  $K \prec_{\Gamma^*} H$  if  $K \prec_T H$
11.  $L \prec_{\Gamma^*} H$  if  $L$  is  $\beta$ -defined

The literal ordering  $\prec_{\Gamma^*}$  is extended to ground clauses by multiset extension and identified with  $\prec_{\Gamma^*}$  as well.

**Lemma 10 (Properties of  $\prec_{\Gamma^*}$ ).**

1.  $\prec_{\Gamma^*}$  is well-defined.
2.  $\prec_{\Gamma^*}$  is a total strict order, i.e.  $\prec_{\Gamma^*}$  is irreflexive, transitive and total.
3.  $\prec_{\Gamma^*}$  is a well-founded ordering.

*Example 11.* Assume a trail  $\Gamma := [a \approx b^{1:C_0 \cdot \sigma_0}, c \approx d^{1:C_1 \cdot \sigma_1}, f(a') \not\approx f(b')^{1:C_2 \cdot \sigma_2}]$ , select KBO as the term ordering  $\prec_T$  where all symbols have weight one and  $a \prec a' \prec b \prec b' \prec c \prec d \prec f$  and a ground term  $\beta := f(f(a))$ . According to the trail induced ordering we have that  $a \approx b \prec_{\Gamma^*} c \approx d \prec_{\Gamma^*} f(a') \not\approx f(b')$  by 9.2. Furthermore we have that

$$a \approx b \prec_{\Gamma^*} a \not\approx b \prec_{\Gamma^*} c \approx d \prec_{\Gamma^*} c \not\approx d \prec_{\Gamma^*} f(a') \not\approx f(b') \prec_{\Gamma^*} f(a') \approx f(b')$$

by 9.3 and 9.4. Now for any literal  $L$  that is  $\beta$ -defined in  $\Gamma$  and the defining literal is  $a \approx b$  it holds that  $a \not\approx b \prec_{\Gamma^*} L \prec_{\Gamma^*} c \approx d$  by 9.6 and 9.7. This holds analogously for all literals that are  $\beta$ -defined in  $\Gamma$  and the defining literal is  $c \approx d$  or  $f(a') \not\approx f(b')$ . Thus we get:

$$\begin{aligned} L_1 \prec_{\Gamma^*} \dots \prec_{\Gamma^*} a \approx b \prec_{\Gamma^*} a \not\approx b \prec_{\Gamma^*} f(a) \approx f(b) \prec_{\Gamma^*} f(a) \not\approx f(b) \prec_{\Gamma^*} \\ c \approx d \prec_{\Gamma^*} c \not\approx d \prec_{\Gamma^*} f(c) \approx f(d) \prec_{\Gamma^*} f(c) \not\approx f(d) \prec_{\Gamma^*} \\ f(a') \not\approx f(b') \prec_{\Gamma^*} f(a') \approx f(b') \prec_{\Gamma^*} a' \approx b' \prec_{\Gamma^*} a' \not\approx b' \prec_{\Gamma^*} K_1 \prec_{\Gamma^*} \dots \end{aligned}$$

where  $K_i$  are the  $\beta$ -undefined literals and  $L_j$  are the trivially defined literals.



**Definition 12 (Rewrite Step).** A rewrite step is a five-tuple  $(s\#t \cdot \sigma, s\#t \vee C \cdot \sigma, R, S, p)$  and inductively defined as follows. The tuple  $(s\#t \cdot \sigma, s\#t \vee C \cdot \sigma, \epsilon, \epsilon, \epsilon)$  is a rewrite step. Given rewrite steps  $R, S$  and a position  $p$  then  $(s\#t \cdot \sigma, s\#t \vee C \cdot \sigma, R, S, p)$  is a rewrite step. The literal  $s\#t$  is called the rewrite literal. In case  $R, S$  are not  $\epsilon$ , the rewrite literal of  $R$  is an equation.

Rewriting is one of the core features of our calculus. The following definition describes a rewrite inference between two clauses. Note that unlike the superposition calculus we allow rewriting below variable level.

**Definition 13 (Rewrite Inference).** Let  $I_1 := (l_1 \approx r_1 \cdot \sigma_1, l_1 \approx r_1 \vee C_1 \cdot \sigma_1, R_1, L_1, p_1)$  and  $I_2 := (l_2 \# r_2 \cdot \sigma_2, l_2 \# r_2 \vee C_2 \cdot \sigma_2, R_2, L_2, p_2)$  be two variable disjoint rewrite steps where  $r_1 \sigma_1 \prec_T l_1 \sigma_1$ ,  $(l_2 \# r_2) \sigma_2|_p = l_1 \sigma_1$  for some position  $p$ . We distinguish two cases:

1. if  $p \in \text{pos}(l_2 \# r_2)$  and  $\mu := \text{mgu}((l_2 \# r_2)|_p, l_1)$  then  $((l_2 \# r_2)[r_1]_p)\mu \cdot \sigma_1 \sigma_2$ ,  $((l_2 \# r_2)[r_1]_p)\mu \vee C_1 \mu \vee C_2 \mu \cdot \sigma_1 \sigma_2, I_1, I_2, p)$  is the result of a rewrite inference.
2. if  $p \notin \text{pos}(l_2 \# r_2)$  then let  $(l_2 \# r_2)\delta$  be the most general instance of  $l_2 \# r_2$  such that  $p \in \text{pos}((l_2 \# r_2)\delta)$ ,  $\delta$  introduces only fresh variables and  $(l_2 \# r_2)\delta \sigma_2 \rho = (l_2 \# r_2) \sigma_2$  for some minimal  $\rho$ . Let  $\mu := \text{mgu}((l_2 \# r_2)\delta|_p, l_1)$ . Then  $((l_2 \# r_2)\delta[r_1]_p \mu \cdot \sigma_1 \sigma_2 \rho, (l_2 \# r_2)\delta[r_1]_p \mu \vee C_1 \mu \vee C_2 \delta \mu \cdot \sigma_1 \sigma_2 \rho, I_1, I_2, p)$  is the result of a rewrite inference.

**Lemma 14.** Let  $I_1 := (l_1 \approx r_1 \cdot \sigma_1, l_1 \approx r_1 \vee C_1 \cdot \sigma_1, R_1, L_1, p_1)$  and  $I_2 := (l_2 \# r_2 \cdot \sigma_2, l_2 \# r_2 \vee C_2 \cdot \sigma_2, R_2, L_2, p_2)$  be two variable disjoint rewrite steps where  $r_1 \sigma_1 \prec_T l_1 \sigma_1$ ,  $(l_2 \# r_2) \sigma_2|_p = l_1 \sigma_1$  for some position  $p$ . Let  $I_3 := (l_3 \# r_3 \cdot \sigma_3, l_3 \# r_3 \vee C_3 \cdot \sigma_3, I_1, I_2, p)$  be the result of a rewrite inference. Then:

1.  $C_3 \sigma_3 = (C_1 \vee C_2) \sigma_1 \sigma_2$  and  $l_3 \# r_3 \sigma_3 = (l_2 \# r_2) \sigma_2 [r_1 \sigma_1]_p$ .
2.  $(l_3 \# r_3) \sigma_3 \prec_T (l_2 \# r_2) \sigma_2$
3. If  $N \models (l_1 \approx r_1 \vee C_1) \wedge (l_2 \# r_2 \vee C_2)$  for some set of clauses  $N$ , then  $N \models l_3 \# r_3 \vee C_3$

Now that we have defined rewrite inferences we can use them to define a *reduction chain application* and a *refutation*, which are sequences of rewrite steps. Intuitively speaking, a *reduction chain application* reduces a literal in a clause with literals in  $\text{conv}(\Gamma)$  until it is irreducible. A *refutation* for a literal  $L$  that is  $\beta$ -false in  $\Gamma$  for a given  $\beta$ , is a sequence of rewrite steps with literals in  $\Gamma, L$  such that  $\perp$  is inferred. Refutations for the literals of the conflict clause will be examined during conflict resolution by the rule Explore-Refutation.

**Definition 15 (Reduction Chain).** Let  $\Gamma$  be a trail. A reduction chain  $\mathcal{P}$  from  $\Gamma$  is a sequence of rewrite steps  $[I_1, \dots, I_m]$  such that for each  $I_i = (s_i \# t_i \cdot \sigma_i, s_i \# t_i \vee C_i \cdot \sigma_i, I_j, I_k, p_i)$  either

1.  $s_i \# t_i^{n_i : s_i \# t_i \vee C_i \cdot \sigma}$  is contained in  $\Gamma$  and  $I_j = I_k = p_i = \epsilon$  or
2.  $I_i$  is the result of a rewriting inference from rewrite steps  $I_j, I_k$  out of  $[I_1, \dots, I_m]$  where  $j, k < i$ .

Let  $(l \# r)^{\delta^{o:l} \# r \vee C \cdot \delta}$  be an annotated ground literal. A reduction chain application from  $\Gamma$  to  $l \# r$  is a reduction chain  $[I_1, \dots, I_m]$  from  $\Gamma$ ,  $(l \# r)^{\delta^{o:l} \# r \vee C \cdot \delta}$  such that  $l\delta \downarrow_{\text{conv}(\Gamma)} = s_m \sigma_m$  and  $r\delta \downarrow_{\text{conv}(\Gamma)} = t_m \sigma_m$ . We assume reduction chain applications to be minimal, i.e., if any rewrite step is removed from the sequence it is no longer a reduction chain application.

**Definition 16 (Refutation).** Let  $\Gamma$  be a trail and  $(l \# r)^{\delta^{o:l} \# r \vee C \cdot \delta}$  an annotated ground literal that is  $\beta$ -false in  $\Gamma$  for a given  $\beta$ . A refutation  $\mathcal{P}$  from  $\Gamma$  and  $l \# r$  is a reduction chain  $[I_1, \dots, I_m]$  from  $\Gamma$ ,  $(l \# r)^{\delta^{o:l} \# r \vee C \cdot \delta}$  such that  $(s_m \# t_m) \sigma_m = s \not\approx s$  for some  $s$ . We assume refutations to be minimal, i.e., if any rewrite step  $I_k$ ,  $k < m$  is removed from the refutation, it is no longer a refutation.

### 3.1 The SCL(EQ) Inference Rules

We can now define the rules of our calculus based on the previous definitions. A *state* is a six-tuple  $(\Gamma; N; U; \beta; k; D)$  similar to the SCL calculus, where  $\Gamma$  a sequence of annotated ground literals,  $N$  and  $U$  the sets of initial and learned clauses,  $\beta$  is a ground term such that for all  $L \in \Gamma$  it holds  $L \prec_T \beta$ ,  $k$  is the decision level, and  $D$  a status that is  $\top$ ,  $\perp$  or a closure  $C \cdot \sigma$ . Before we propagate or decide any literal, we make sure that it is irreducible in the current trail. Together with the design of  $\prec_{\Gamma^*}$  this eventually enables rewriting as a simplification rule.

#### Propagate

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\text{SCL(EQ)}} (\Gamma, s_m \# t_m \sigma_m^{k:(s_m \# t_m \vee C_m) \cdot \sigma_m}; N; U; \beta; k; \top)$   
provided there is a  $C \in (N \cup U)$ ,  $\sigma$  grounding for  $C$ ,  $C = C_0 \vee C_1 \vee L$ ,  $\Gamma \models \neg C_0 \sigma$ ,  $C_1 \sigma = L \sigma \vee \dots \vee L \sigma$ ,  $C_1 = L_1 \vee \dots \vee L_n$ ,  $\mu = \text{mgu}(L_1, \dots, L_n, L)$   $L \sigma$  is  $\beta$ -undefined in  $\Gamma$ ,  $(C_0 \vee L) \mu \sigma \prec_T \beta$ ,  $\sigma$  is irreducible by  $\text{conv}(\Gamma)$ ,  $[I_1, \dots, I_m]$  is a reduction chain application from  $\Gamma$  to  $L \sigma^{k:(L \vee C_0) \mu \cdot \sigma}$  where  $I_m = (s_m \# t_m \cdot \sigma_m, s_m \# t_m \vee C_m \cdot \sigma_m, I_j, I_k, p_m)$ .

Note that the definition of Propagate also includes the case where  $L \sigma$  is irreducible by  $\Gamma$ . In this case  $L = s_m \# t_m$  and  $m = 1$ . The rule Decide below, is similar to Propagate, except for the subclause  $C_0$  which must be  $\beta$ -undefined or  $\beta$ -true in  $\Gamma$ , i.e., Propagate cannot be applied and the decision literal is annotated by a tautology.

#### Decide

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\text{SCL(EQ)}} (\Gamma, s_m \# t_m \sigma_m^{k+1:(s_m \# t_m \vee \text{comp}(s_m \# t_m)) \cdot \sigma_m}; N; U; \beta; k+1; \top)$   
provided there is a  $C \in (N \cup U)$ ,  $\sigma$  grounding for  $C$ ,  $C = C_0 \vee L$ ,  $C_0 \sigma$  is  $\beta$ -undefined or  $\beta$ -true in  $\Gamma$ ,  $L \sigma$  is  $\beta$ -undefined in  $\Gamma$ ,  $(C_0 \vee L) \sigma \prec_T \beta$ ,  $\sigma$  is irreducible by  $\text{conv}(\Gamma)$ ,  $[I_1, \dots, I_m]$  is a reduction chain application from  $\Gamma$  to  $L \sigma^{k+1:L \vee C_0 \cdot \sigma}$  where  $I_m = (s_m \# t_m \cdot \sigma_m, s_m \# t_m \vee C_m \cdot \sigma_m, I_j, I_k, p_m)$ .

**Conflict**

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\text{SCL}(\text{EQ})} (\Gamma; N; U; \beta; k; D)$   
 provided there is a  $D' \in (N \cup U)$ ,  $\sigma$  grounding for  $D'$ ,  $D'\sigma$  is  $\beta$ -false in  $\Gamma$ ,  $\sigma$  is irreducible by  $\text{conv}(\Gamma)$ ,  $D = \perp$  if  $D'\sigma$  is of level 0 and  $D = D' \cdot \sigma$  otherwise.

For the non-equational case, when a conflict clause is found by an SCL calculus [14, 18], the complements of its first-order ground literals are contained in the trail. For equational literals this is not the case, in general. The proof showing  $D$  to be  $\beta$ -false with respect to  $\Gamma$  is a rewrite proof with respect to  $\text{conv}(\Gamma)$ . This proof needs to be analyzed to eventually perform paramodulation steps on  $D$  or to replace  $D$  by a  $\prec_{\Gamma^*}$  smaller  $\beta$ -false clause showing up in the proof.

**Skip**

$(\Gamma, K^{l:C \cdot \tau}, L^{k:C' \cdot \tau'}; N; U; \beta; k; D \cdot \sigma) \Rightarrow_{\text{SCL}(\text{EQ})} (\Gamma, K^{l:C \cdot \tau}; N; U; \beta; l; D \cdot \sigma)$  if  $D\sigma$  is  $\beta$ -false in  $\Gamma, K^{l:C \cdot \tau}$ .

The Explore-Refutation rule is the FOL with Equality counterpart to the resolve rule in CDCL or SCL. While in CDCL or SCL complementary literals of the conflict clause are present on the trail and can directly be used for resolution steps, this needs a generalization for FOL with Equality. Here, in general, we need to look at (rewriting) refutations of the conflict clause and pick an appropriate clause from the refutation as the next conflict clause.

**Explore-Refutation**

$(\Gamma, L; N; U; \beta; k; (D \vee s \# t) \cdot \sigma) \Rightarrow_{\text{SCL}(\text{EQ})} (\Gamma, L; N; U; \beta; k; (s_j \# t_j \vee C_j) \cdot \sigma_j)$   
 if  $(s \# t)\sigma$  is strictly  $\prec_{\Gamma^*}$  maximal in  $(D \vee s \# t)\sigma$ ,  $L$  is the defining literal of  $(s \# t)\sigma$ ,  $[I_1, \dots, I_m]$  is a refutation from  $\Gamma$  and  $(s \# t)\sigma$ ,  $I_j = (s_j \# t_j \cdot \sigma_j, (s_j \# t_j \vee C_j) \cdot \sigma_j, I_l, I_k, p_j)$ ,  $1 \leq j \leq m$ ,  $(s_j \# t_j \vee C_j)\sigma_j \prec_{\Gamma^*} (D \vee s \# t)\sigma$ ,  $(s_j \# t_j \vee C_j)\sigma_j$  is  $\beta$ -false in  $\Gamma$ .

**Factorize**

$(\Gamma; N; U; \beta; k; (D \vee L \vee L') \cdot \sigma) \Rightarrow_{\text{SCL}(\text{EQ})} (\Gamma; N; U; \beta; k; (D \vee L)\mu \cdot \sigma)$   
 provided  $L\sigma = L'\sigma$ , and  $\mu = \text{mgu}(L, L')$ .

**Equality-Resolution**

$(\Gamma; N; U; \beta; k; (D \vee s \not\approx s') \cdot \sigma) \Rightarrow_{\text{SCL}(\text{EQ})} (\Gamma; N; U; \beta; k; D\mu \cdot \sigma)$   
 provided  $s\sigma = s'\sigma$ ,  $\mu = \text{mgu}(s, s')$ .

**Backtrack**

$(\Gamma, K, \Gamma'; N; U; \beta; k; (D \vee L) \cdot \sigma) \Rightarrow_{\text{SCL}(\text{EQ})} (\Gamma; N; U \cup \{D \vee L\}; \beta; j - i; \top)$   
 provided  $D\sigma$  is of level  $i'$  where  $i' < k$ ,  $K$  is of level  $j$  and  $\Gamma, K$  the minimal trail subsequence such that there is a grounding substitution  $\tau$  with  $(D \vee L)\tau$   $\beta$ -false in  $\Gamma, K$  but not in  $\Gamma$ ;  $i = 1$  if  $K$  is a decision literal and  $i = 0$  otherwise.

**Grow**

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\text{SCL}(\text{EQ})} (\epsilon; N; U; \beta'; 0; \top)$   
 provided  $\beta \prec_T \beta'$ .

In addition to soundness and completeness of the SCL(EQ) rules their tractability in practice is an important property for a successful implementation. In particular, finding propagating literals or detecting a false clause under some grounding. It turns out that these operations are NP-complete, similar to first-order subsumption which has been shown to be tractable in practice.

**Lemma 17.** *Assume that all ground terms  $t$  with  $t \prec_T \beta$  for any  $\beta$  are polynomial in the size of  $\beta$ . Then testing Propagate (Conflict) is NP-Complete, i.e., the problem of checking for a given clause  $C$  whether there exists a grounding substitution  $\sigma$  such that  $C\sigma$  propagates (is false) is NP-Complete.*

*Example 18 (SCL(EQ) vs. Superposition: Saturation).* Consider the following clauses:

$$N := \{C_1 := c \approx d \vee D, C_2 := a \approx b \vee c \not\approx d, C_3 := f(a) \not\approx f(b) \vee g(c) \not\approx g(d)\}$$

where again we assume a KBO with all symbols having weight one, precedence  $d \prec c \prec b \prec a \prec g \prec f$  and  $\beta := f(f(g(a)))$ . Suppose that we first decide  $c \approx d$  and then propagate  $a \approx b$ :  $\Gamma = [c \approx d^{1:c \approx d \vee c \not\approx d}, a \approx b^{1:C_2}]$ . Now we have a conflict with  $C_3$ . Explore-Refutation applied to the conflict clause  $C_3$  results in a paramodulation inference between  $C_3$  and  $C_2$ . Another application of Equality-Resolution gives us the new conflict clause  $C_4 := c \not\approx d \vee g(c) \not\approx g(d)$ . Now we can Skip the last literal on the trail, which gives us  $\Gamma = [c \approx d^{1:c \approx d \vee c \not\approx d}]$ . Another application of the Explore-Refutation rule to  $C_4$  using the decision justification clause followed by Equality-Resolution and Factorize gives us  $C_5 := c \not\approx d$ . Thus with SCL(EQ) the following clauses remain:

$$\begin{aligned} C'_1 &= D & C_5 &= c \not\approx d \\ C_3 &= f(a) \not\approx f(b) \vee g(c) \not\approx g(d) \end{aligned}$$

where we derived  $C'_1$  out of  $C_1$  by subsumption resolution [33] using  $C_5$ . Actually, subsumption resolution is compatible with the general redundancy notion of SCL(EQ), see Lemma 25. Now we consider the same example with superposition and the very same ordering ( $N_i$  is the clause set of the previous step and  $N_0$  the initial clause set  $N$ ).

$$\begin{aligned} N_0 &\Rightarrow_{\text{Sup}(C_2, C_3)} N_1 \cup \{C_4 := c \not\approx d \vee g(c) \not\approx g(d)\} \\ &\Rightarrow_{\text{Sup}(C_1, C_4)} N_2 \cup \{C_5 := c \not\approx d \vee D\} \Rightarrow_{\text{Sup}(C_1, C_5)} N_3 \cup \{C_6 := D\} \end{aligned}$$

Thus superposition ends up with the following clauses:

$$\begin{aligned} C_2 &= a \approx b \vee c \not\approx d & C_3 &= f(a) \not\approx f(b) \vee g(c) \not\approx g(d) \\ C_4 &= c \not\approx d \vee g(c) \not\approx g(d) & C_6 &= D \end{aligned}$$

The superposition calculus generates more and larger clauses.

*Example 19 (SCL(EQ) vs. Superposition: Refutation).* Suppose the following set of clauses:  $N := \{C_1 := f(x) \not\approx a \vee f(x) \approx b, C_2 := f(f(y)) \approx y, C_3 := a \not\approx b\}$  where again we assume a KBO with all symbols having weight one, precedence

$b \prec a \prec f$  and  $\beta := f(f(f(a)))$ . A long refutation by the superposition calculus results in the following ( $N_i$  is the clause set of the previous step and  $N_0$  the initial clause set  $N$ ):

$$\begin{aligned}
N_0 &\Rightarrow_{Sup(C_1, C_2)} N_1 \cup \{C_4 := y \not\approx a \vee f(f(y)) \approx b\} \\
&\Rightarrow_{Sup(C_1, C_4)} N_2 \cup \{C_5 := a \not\approx b \vee f(f(y)) \approx b \vee y \not\approx a\} \\
&\Rightarrow_{Sup(C_2, C_5)} N_3 \cup \{C_6 := a \not\approx b \vee b \approx y \vee y \not\approx a\} \\
&\Rightarrow_{Sup(C_2, C_4)} N_4 \cup \{C_7 := y \approx b \vee y \not\approx a\} \\
&\Rightarrow_{EqRes(C_7)} N_5 \cup \{C_8 := a \approx b\} \Rightarrow_{Sup(C_3, C_8)} N_6 \cup \{\perp\}
\end{aligned}$$

The shortest refutation by the superposition calculus is as follows:

$$\begin{aligned}
N_0 &\Rightarrow_{Sup(C_1, C_2)} N_1 \cup \{C_4 := y \not\approx a \vee f(f(y)) \approx b\} \\
&\Rightarrow_{Sup(C_2, C_4)} N_2 \cup \{C_5 := y \approx b \vee y \not\approx a\} \\
&\Rightarrow_{EqRes(C_5)} N_3 \cup \{C_6 := a \approx b\} \Rightarrow_{Sup(C_3, C_6)} N_4 \cup \{\perp\}
\end{aligned}$$

In SCL(EQ) on the other hand we would always first propagate  $a \not\approx b$ ,  $f(f(a)) \approx a$  and  $f(f(b)) \approx b$ . As soon as  $a \not\approx b$  and  $f(f(a)) \approx a$  are propagated we have a conflict with  $C_1\{x \rightarrow f(a)\}$ . So suppose in the worst case we propagate:

$$\Gamma := [a \not\approx b^{0:a \not\approx b}, f(f(b)) \approx b^{0:(f(f(y)) \approx y)\{y \rightarrow b\}}, f(f(a)) \approx a^{0:(f(f(y)) \approx y)\{y \rightarrow a\}}]$$

Now we have a conflict with  $C_1\{x \rightarrow f(a)\}$ . Since there is no decision literal on the trail, *Conflict* rule immediately returns  $\perp$  and we are done.

## 4 Soundness and Completeness

In this section we show soundness and refutational completeness of SCL(EQ) under the assumption of a regular run. We provide the definition of a regular run and show that for a regular run all learned clauses are non-redundant according to our trail induced ordering. We start with the definition of a sound state.

**Definition 20.** A state  $(\Gamma; N; U; \beta; k; D)$  is sound if the following conditions hold:

1.  $\Gamma$  is a consistent sequence of annotated literals,
2. for each decomposition  $\Gamma = \Gamma_1, L\sigma^{i:(C \vee L) \cdot \sigma}, \Gamma_2$  where  $L\sigma$  is a propagated literal, we have that  $C\sigma$  is  $\beta$ -false in  $\Gamma_1$ ,  $L\sigma$  is  $\beta$ -undefined in  $\Gamma_1$  and irreducible by  $\text{conv}(\Gamma_1)$ ,  $N \cup U \models (C \vee L)$  and  $(C \vee L)\sigma \prec_T \beta$ ,
3. for each decomposition  $\Gamma = \Gamma_1, L\sigma^{i:(L \vee \text{comp}(L)) \cdot \sigma}, \Gamma_2$  where  $L\sigma$  is a decision literal, we have that  $L\sigma$  is  $\beta$ -undefined in  $\Gamma_1$  and irreducible by  $\text{conv}(\Gamma_1)$ ,  $N \cup U \models (L \vee \text{comp}(L))$  and  $(L \vee \text{comp}(L))\sigma \prec_T \beta$ ,
4.  $N \models U$ ,
5. if  $D = C \cdot \sigma$ , then  $C\sigma$  is  $\beta$ -false in  $\Gamma$ ,  $N \cup U \models C$ ,

**Lemma 21.** The initial state  $(\epsilon; N; \emptyset; \beta; 0; \top)$  is sound.

**Definition 22.** A run is a sequence of applications of SCL(EQ) rules starting from the initial state.

**Theorem 23.** *Assume a state  $(\Gamma; N; U; \beta; k; D)$  resulting from a run. Then  $(\Gamma; N; U; \beta; k; D)$  is sound.*

Next, we give the definition of a regular run. Intuitively speaking, in a regular run we are always allowed to do decisions except if

1. a literal can be propagated before the first decision and
2. the negation of a literal can be propagated.

To ensure non-redundant learning we enforce at least one application of Skip during conflict resolution except for the special case of a conflict after a decision.

**Definition 24 (Regular Run).** *A run is called regular if*

1. *the rules Conflict and Factorize have precedence over all other rules,*
2. *If  $k = 0$  in a state  $(\Gamma; N; U; \beta; k; D)$ , then Propagate has precedence over Decide,*
3. *If an annotated literal  $L^{k:C \cdot \sigma}$  could be added by an application of Propagate on  $\Gamma$  in a state  $(\Gamma; N; U; \beta; k; D)$  and  $C \in N \cup U$ , then the annotated literal  $\text{comp}(L)^{k+1:C' \cdot \sigma'}$  is not added by Decide on  $\Gamma$ ,*
4. *during conflict resolution Skip is applied at least once, except if Conflict is applied immediately after an application of Decide.*
5. *if Conflict is applied immediately after an application of Decide, then Backtrack is only applied in a state  $(\Gamma, L'; N; U; \beta; k; D \cdot \sigma)$  if  $L\sigma = \text{comp}(L')$  for some  $L \in D$ .*

Now we show that any learned clause in a regular run is non-redundant according to our trail induced ordering.

**Lemma 25 (Non-Redundant Clause Learning).** *Let  $N$  be a clause set. The clauses learned during a regular run in SCL(EQ) are not redundant with respect to  $\prec_{\Gamma^*}$  and  $N \cup U$ . For the trail only non-redundant clauses need to be considered.*

The proof of Lemma 25 is based on the fact that conflict resolution eventually produces a clause smaller than the original conflict clause with respect to  $\prec_{\Gamma^*}$ . All simplifications, e.g., contextual rewriting, as defined in [2, 20, 33, 35–37], are therefore compatible with Lemma 25 and may be applied to the newly learned clause as long as they respect the induced trail ordering. In detail, let  $\Gamma$  be the trail before the application of rule Backtrack. The newly learned clause can be simplified according to the induced trail ordering  $\prec_{\Gamma^*}$  as long as the simplified clause is smaller with respect to  $\prec_{\Gamma^*}$ .

Another important consequence of Lemma 25 is that newly learned clauses need not to be considered for redundancy. Furthermore, the SCL(EQ) calculus always terminates, Lemma 33, because there only finitely many non-redundant clauses with respect to a fixed  $\beta$ .

For dynamic redundancy, we have to consider the fact that the induced trail ordering changes. At this level, only redundancy criteria and simplifications that

are compatible with *all* induced trail orderings may be applied. Due to the construction of the induced trail ordering, it is compatible with  $\prec_T$  for unit clauses.

**Lemma 26 (Unit Rewriting).** *Assume a state  $(\Gamma; N; U; \beta; k; D)$  resulting from a regular run where the current level  $k > 0$  and a unit clause  $l \approx r \in N$ . Now assume a clause  $C \vee L[l']_p \in N$  such that  $l' = l\mu$  for some matcher  $\mu$ . Now assume some arbitrary grounding substitutions  $\sigma'$  for  $C \vee L[l']_p$ ,  $\sigma$  for  $l \approx r$  such that  $l\sigma = l'\sigma'$  and  $r\sigma \prec_T l\sigma$ . Then  $(C \vee L[r\mu\sigma\sigma']_p)\sigma' \prec_{\Gamma^*} (C \vee L[l']_p)\sigma'$ .*

In addition, any notion that is based on a literal subset relationship is also compatible with ordering changes. The standard example is subsumption.

**Lemma 27.** *Let  $C, D$  be two clauses. If there exists a substitution  $\sigma$  such that  $C\sigma \subset D$ , then  $D$  is redundant with respect to  $C$  and any  $\prec_{\Gamma^*}$ .*

The notion of redundancy, Definition 1, only supports a strict subset relation for Lemma 27, similar to the superposition calculus. However, the newly generated clauses of SCL(EQ) are the result of paramodulation inferences [28]. In a recent contribution to dynamic, abstract redundancy [32] it is shown that also the non-strict subset relation in Lemma 27, i.e.,  $C\sigma \subseteq D$ , preserves completeness.

If all stuck states, see below Definition 28, with respect to a fixed  $\beta$  are visited before increasing  $\beta$  then this provides a simple dynamic fairness strategy.

When unit reduction or any other form of supported rewriting is applied to clauses smaller than the current  $\beta$ , it can be applied independently from the current trail. If, however, unit reduction is applied to clauses larger than the current  $\beta$  then the calculus must do a restart to its initial state, in particular the trail must be emptied, as for otherwise rewriting may result generating a conflict that did not exist with respect to the current trail before the rewriting. This is analogous to a restart in CDCL once a propositional unit clause is derived and used for simplification. More formally, we add the following new Restart rule to the calculus to reset the trail to its initial state after a unit reduction.

### Restart

$(\Gamma; N; U; \beta; k; \top) \Rightarrow_{\text{SCL(EQ)}} (\epsilon; N; U; \beta; 0; \top)$

Next we show refutation completeness of SCL(EQ). To achieve this we first give a definition of a stuck state. Then we show that stuck states only occur if all ground literals  $L \prec_T \beta$  are  $\beta$ -defined in  $\Gamma$  and not during conflict resolution. Finally we show that conflict resolution will always result in an application of Backtrack. This allows us to show termination (without application of Grow) and refutational completeness.

**Definition 28 (Stuck State).** *A state  $(\Gamma; N; U; \beta; k; D)$  is called stuck if  $D \neq \perp$  and none of the rules of the calculus, except for Grow, is applicable.*

**Lemma 29 (Form of Stuck States).** *If a regular run (without rule Grow) ends in a stuck state  $(\Gamma; N; U; \beta; k; D)$ , then  $D = \top$  and all ground literals  $L\sigma \prec_T \beta$ , where  $L \vee C \in (N \cup U)$  are  $\beta$ -defined in  $\Gamma$ .*

**Lemma 30.** *Suppose a sound state  $(\Gamma; N; U; \beta; k; D)$  resulting from a regular run where  $D \notin \{\top, \perp\}$ . If Backtrack is not applicable then any set of applications of Explore-Refutation, Skip, Factorize, Equality-Resolution will finally result in a sound state  $(\Gamma'; N; U; \beta; k; D')$ , where  $D' \prec_{\Gamma^*} D$ . Then Backtrack will be finally applicable.*

**Corollary 31 (Satisfiable Clause Sets).** *Let  $N$  be a satisfiable clause set. Then any regular run without rule Grow will end in a stuck state, for any  $\beta$ .*

Thus a stuck state can be seen as an indication for a satisfiable clause set. Of course, it remains to be investigated whether the clause set is actually satisfiable. Superposition is one of the strongest approaches to detect satisfiability and constitutes a decision procedure for many decidable first-order fragments [4, 19]. Now given a stuck state and some specific ordering such as KBO, LPO, or some polynomial ordering [17], it is decidable whether the ordering can be instantiated from a stuck state such that  $\Gamma$  coincides with the superposition model operator on the ground terms smaller than  $\beta$ . In this case it can be effectively checked whether the clauses derived so far are actually saturated by the superposition calculus with respect to this specific ordering. In this sense, SCL(EQ) has the same power to decide satisfiability of first-order clause sets than superposition.

**Definition 32.** *A regular run terminates in a state  $(\Gamma; N; U; \beta; k; D)$  if  $D = \top$  and no rule is applicable, or  $D = \perp$ .*

**Lemma 33.** *Let  $N$  be a set of clauses and  $\beta$  be a ground term. Then any regular run that never uses Grow terminates.*

**Lemma 34.** *If a regular run reaches the state  $(\Gamma; N; U; \beta; k; \perp)$  then  $N$  is unsatisfiable.*

**Theorem 35 (Refutational Completeness).** *Let  $N$  be an unsatisfiable clause set, and  $\prec_T$  a desired term ordering. For any ground term  $\beta$  where  $\text{gnd}_{\prec_T \beta}(N)$  is unsatisfiable, any regular SCL(EQ) run without rule Grow will terminate by deriving  $\perp$ .*

## 5 Discussion

We presented SCL(EQ), a new sound and complete calculus for reasoning in first-order logic with equality. We will now discuss some of its aspects and present ideas for future work beyond the scope of this paper.

The trail induced ordering, Definition 9, is the result of letting the calculus follow the logical structure of the clause set on the literal level and at the same time supporting rewriting at the term level. It can already be seen by examples on ground clauses over (in)equations over constants that this combination requires a layered approach as suggested by Definition 9, see [24].

In case the calculus runs into a stuck state, i.e., the current trail is a model for the set of considered ground instances, then the trail information can be



effectively used for a guided continuation. For example, in order to use the trail to certify a model, the trail literals can be used to guide the design of a lifted ordering for the clauses with variables such that propagated trail literals are maximal in respective clauses. Then it could be checked by superposition, if the current clause is saturated by such an ordering. If this is not the case, then there must be a superposition inference larger than the current  $\beta$ , thus giving a hint on how to extend  $\beta$ . Another possibility is to try to extend the finite set of ground terms considered in a stuck state to the infinite set of all ground terms by building extended equivalence classes following patterns that ensure decidability of clause testing, similar to the ideas in [14]. If this fails, then again this information can be used to find an appropriate extension term  $\beta$  for rule Grow.

In contrast to superposition, SCL(EQ) does also inferences below variable level. Inferences in SCL(EQ) are guided by a false clause with respect to a partial model assumption represented by the trail. Due to this guidance and the different style of reasoning this does not result in an explosion in the number of possibly inferred clauses but also rather in the derivation of more general clauses, see [24].

Currently, the reasoning with solely positive equations is done on and with respect to the trail. It is well-known that also inferences from this type of reasoning can be used to speed up the overall reasoning process. The SCL(EQ) calculus already provides all information for such a type of reasoning, because it computes the justification clauses for trail reasoning via rewriting inferences. By an assessment of the quality of these clauses, e.g., their reduction potential with respect to trail literals, they could also be added, independently from resolving a conflict.

The trail reasoning is currently defined with respect to rewriting. It could also be performed by congruence closure [26].

Towards an implementation, the aspect of how to find interesting ground decision or propagation literals for the trail can be treated similar to CDCL [11, 21, 25, 29]. A simple heuristic may be used from the start, like counting the number of instance relationships of some ground literal with respect to the clause set, but later on a bonus system can focus the search towards the structure of the clause sets. Ground literals involved in a conflict or the process of learning a new clause get a bonus or preference. The regular strategy requires the propagation of all ground unit clauses smaller than  $\beta$ . For an implementation a propagation of the (explicit and implicit) unit clauses with variables to the trail will be a better choice. This complicates the implementation of refutation proofs and rewriting (congruence closure), but because every reasoning is layered by a ground term  $\beta$  this can still be efficiently done.

**Acknowledgments.** This work was partly funded by DFG grant 389792660 as part of TRR 248, see <https://perspicuous-computing.science>. We thank the anonymous reviewers and Martin Desharnais for their thorough reading, detailed comments, and corrections.

## References

1. Alagi, G., Weidenbach, C.: NRCL - a model building approach to the Bernays-Schönfinkel fragment. In: Lutz, C., Ranise, S. (eds.) FroCoS 2015. LNCS (LNAI), vol. 9322, pp. 69–84. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-24246-0\\_5](https://doi.org/10.1007/978-3-319-24246-0_5)
2. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.* **4**(3), 217–247 (1994)
3. Bachmair, L., Ganzinger, H., Voronkov, A.: Elimination of equality via transformation with ordering constraints. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS, vol. 1421, pp. 175–190. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0054259>
4. Bachmair, L., Ganzinger, H., Waldmann, U.: Superposition with simplification as a decision procedure for the monadic class with equality. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) KGC 1993. LNCS, vol. 713, pp. 83–96. Springer, Heidelberg (1993). <https://doi.org/10.1007/BFb0022557>
5. Baumgartner, P.: Hyper tableau — the next generation. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 60–76. Springer, Heidelberg (1998). [https://doi.org/10.1007/3-540-69778-0\\_14](https://doi.org/10.1007/3-540-69778-0_14)
6. Baumgartner, P., Fuchs, A., Tinelli, C.: Lemma learning in the model evolution calculus. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 572–586. Springer, Heidelberg (2006). [https://doi.org/10.1007/11916277\\_39](https://doi.org/10.1007/11916277_39)
7. Baumgartner, P., Furbach, U., Pelzer, B.: Hyper tableaux with equality. In: Pfennig, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 492–507. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73595-3\\_36](https://doi.org/10.1007/978-3-540-73595-3_36)
8. Baumgartner, P., Pelzer, B., Tinelli, C.: Model evolution with equality-revised and implemented. *J. Symb. Comput.* **47**(9), 1011–1045 (2012)
9. Baumgartner, P., Tinelli, C.: The model evolution calculus with equality. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 392–408. Springer, Heidelberg (2005). [https://doi.org/10.1007/11532231\\_29](https://doi.org/10.1007/11532231_29)
10. Baumgartner, P., Waldmann, U.: Superposition and model evolution combined. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 17–34. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02959-2\\_2](https://doi.org/10.1007/978-3-642-02959-2_2)
11. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
12. Bonacina, M.P., Furbach, U., Sofronie-Stokkermans, V.: On First-Order Model-Based Reasoning. In: Martí-Oliet, N., Ölveczky, P.C., Talcott, C. (eds.) Logic, Rewriting, and Concurrency. LNCS, vol. 9200, pp. 181–204. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-23165-5\\_8](https://doi.org/10.1007/978-3-319-23165-5_8)
13. Bonacina, M.P., Plaisted, D.A.: SGGs theorem proving: an exposition. In: Schulz, S., Moura, L.D., Konev, B. (eds.) PAAR-2014. 4th Workshop on Practical Aspects of Automated Reasoning. EPiC Series in Computing, vol. 31, pp. 25–38. EasyChair (2015)
14. Bromberger, M., Fiori, A., Weidenbach, C.: Deciding the Bernays-Schoenfinkel fragment over bounded difference constraints by simple clause learning over theories. In: Henglein, F., Shoham, S., Vize, Y. (eds.) VMCAI 2021. LNCS, vol. 12597, pp. 511–533. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67067-2\\_23](https://doi.org/10.1007/978-3-030-67067-2_23)

15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. *Commun. ACM* **5**(7), 394–397 (1962)
16. Davis, M., Putnam, H.: A computing procedure for quantification theory. *J. ACM (JACM)* **7**(3), 201–215 (1960)
17. Dershowitz, N., Plaisted, D.A.: Rewriting. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. I, chap. 9, pp. 535–610. Elsevier (2001)
18. Fiori, A., Weidenbach, C.: SCL clause learning from simple models. In: Fontaine, P. (ed.) *CADE 2019. LNCS (LNAI)*, vol. 11716, pp. 233–249. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-29436-6\\_14](https://doi.org/10.1007/978-3-030-29436-6_14)
19. Ganzinger, H., de Nivelle, H.: A superposition decision procedure for the guarded fragment with equality. In: *LICS*, pp. 295–304 (1999)
20. Gleiss, B., Kovács, L., Rath, J.: Subsumption demodulation in first-order theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) *IJCAR 2020. LNCS (LNAI)*, vol. 12166, pp. 297–315. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51074-9\\_17](https://doi.org/10.1007/978-3-030-51074-9_17)
21. Bayardo, R.J., Schrag, R.: Using CSP look-back techniques to solve exceptionally hard SAT instances. In: Freuder, E.C. (ed.) *CP 1996. LNCS*, vol. 1118, pp. 46–60. Springer, Heidelberg (1996). [https://doi.org/10.1007/3-540-61551-2\\_65](https://doi.org/10.1007/3-540-61551-2_65)
22. Korovin, K.: Inst-Gen – a modular approach to instantiation-based automated reasoning. In: Voronkov, A., Weidenbach, C. (eds.) *Programming Logics. LNCS*, vol. 7797, pp. 239–270. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-37651-1\\_10](https://doi.org/10.1007/978-3-642-37651-1_10)
23. Korovin, K., Stickels, C.: iProver-Eq: an instantiation-based theorem prover with equality. In: Giesl, J., Hähnle, R. (eds.) *IJCAR 2010. LNCS (LNAI)*, vol. 6173, pp. 196–202. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14203-1\\_17](https://doi.org/10.1007/978-3-642-14203-1_17)
24. Leidinger, H., Weidenbach, C.: SCL(EQ): SCL for first-order logic with equality (2022). [arXiv: 2205.08297](https://arxiv.org/abs/2205.08297)
25. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: *Proceedings of the Design Automation Conference*, pp. 530–535. ACM (2001)
26. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. *J. ACM* **27**(2), 356–364 (1980)
27. Plaisted, D.A., Zhu, Y.: Ordered semantic hyper-linking. *J. Autom. Reason.* **25**(3), 167–217 (2000)
28. Robinson, G., Wos, L.: Paramodulation and theorem-proving in first-order theories with equality. In: Meltzer, B., Michie, D. (eds.) *Machine Intelligence 4*, pp. 135–150 (1969)
29. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: *International Conference on Computer Aided Design, ICCAD*, pp. 220–227. IEEE Computer Society Press (1996)
30. Sutcliffe, G.: The TPTP problem library and associated infrastructure - from CNF to th0, TPTP v6.4.0. *J. Autom. Reasoning* **59**(4), 483–502 (2017)
31. Teucle, A.: An approximation and refinement approach to first-order automated reasoning. Doctoral thesis, Saarland University (2018)
32. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) *IJCAR 2020. LNCS (LNAI)*, vol. 12166, pp. 316–334. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51074-9\\_18](https://doi.org/10.1007/978-3-030-51074-9_18)

33. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. 2, chap. 27, pp. 1965–2012. Elsevier (2001)
34. Weidenbach, C.: Automated reasoning building blocks. In: Meyer, R., Platzer, A., Wehrheim, H. (eds.) *Correct System Design*. LNCS, vol. 9360, pp. 172–188. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-23506-6\\_12](https://doi.org/10.1007/978-3-319-23506-6_12)
35. Weidenbach, C., Wischniewski, P.: Contextual rewriting in SPASS. In: PAAR/ESHOL. *CEUR Workshop Proceedings*, vol. 373, pp. 115–124. Australien, Sydney (2008)
36. Weidenbach, C., Wischniewski, P.: Subterm contextual rewriting. *AI Commun.* **23**(2–3), 97–109 (2010)
37. Wischniewski, P.: *Efficient Reasoning Procedures for Complex First-Order Theories*. Ph.D. thesis, Saarland University, November 2012


**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Term Orderings for Non-reachability of (Conditional) Rewriting

Akihisa Yamada<sup>(✉)</sup>

National Institute of Advanced Industrial Science and Technology, Tokyo, Japan  
 akihisa.yamada@aist.go.jp

**Abstract.** We propose generalizations of reduction pairs, well-established techniques for proving termination of term rewriting, in order to prove unsatisfiability of reachability (infeasibility) in plain and conditional term rewriting. We adapt the weighted path order, a merger of the Knuth–Bendix order and the lexicographic path order, into the proposed framework. The proposed approach is implemented in the termination prover NaTT, and the strength of our approach is demonstrated through examples and experiments.

## 1 Introduction

In the research area of term rewriting, among the most well-studied topics are termination, confluence, and reachability analyses.

In termination analysis, a crucial task used to be to design *reduction orders*, well-founded orderings over terms that are closed under contexts and substitutions. Well-known examples of such orderings include the *Knuth–Bendix ordering* [14], *polynomial interpretations* [18], *multiset/lexicographic path ordering* [4, 13], and *matrix interpretations* [5]. The *dependency pair framework* generalized reduction orders into *reduction pairs* [2, 9, 12], and there are a number of implementations that automatically find reduction pairs, e.g., AProVE [7], T<sub>1</sub>T<sub>2</sub> [16], MU-TERM [11], NaTT [35], competing in the International Termination Competition [8].

Traditional reachability analysis (cf. [6]) has been concerned with the possibility of rewriting a given source term  $s$  to a target  $t$ , where variables in the terms are treated as constants. There is an increasing need for solving a more general question: is it possible to instantiate variables so that the instance of  $s$  rewrites to the instance of  $t$ ? Let us illustrate the problem with an elementary example.

*Example 1.* Consider the following TRS encoding addition of natural numbers:

$$\mathcal{R}_{\text{add}} := \{ \text{add}(0, y) \rightarrow y, \text{add}(s(x), y) \rightarrow s(\text{add}(x, y)) \}$$

The reachability constraint  $\text{add}(s(x), y) \rightarrow y$  represents the possibility of rewriting from  $\text{add}(s(x), y)$  to  $y$ , where variables  $x$  and  $y$  can be arbitrary terms.

This (un)satisfiability problem of reachability, also called (in)feasibility, plays important roles in termination [24] and confluence analyses of (conditional) rewriting [21]. A tool competition dedicated for this problem has been founded as the infeasibility (INF) category in the International Confluence Competition (CoCo) since 2019 [25].

In this paper, we propose a new method for proving unsatisfiability of reachability, using the term ordering techniques developed for termination analysis. Specifically, in Sect. 3, we first generalize reduction pairs to *rewrite pairs*, and show that they can be used for proving unsatisfiability of reachability. We further generalize the notion to *co-rewrite pairs*, yielding a sound and complete method. The power of the proposed method is demonstrated by importing (relaxed) *semantic* term orderings from termination analysis.

In order to import also *syntactic* term orderings, in Sect. 4 we identify a condition when the *weighted path order (WPO)* [36] forms a rewrite pair. Since KBO and LPO are instances of WPO, we see that these orderings can also be used in our method. In Sect. 5 we also present how to derive co-rewrite pairs from WPO.

In Sect. 6, we adapt the approach into conditional rewriting. Section 7 reports on the implementation and experiments conducted on examples in the paper and the benchmark set of CoCo 2021.

*Related Work* Our rewrite pairs are essentially Aoto’s *discrimination pairs* [1] which are closed under substitutions. On way of disproving confluence, Aoto introduced discrimination pairs and used them in proving non-joinability. The *joinability* of terms  $s$  and  $t$  is expressed as  $\exists u. s \rightarrow_{\mathcal{R}}^* u \leftarrow_{\mathcal{R}}^* t$ , while the current paper is concerned with  $\exists \theta. s\theta \rightarrow_{\mathcal{R}}^* t\theta$ . As substitutions are not considered, discrimination pairs do not need closure under substitutions, and Aoto’s insights are mainly for dealing with the reverse rewriting  $\leftarrow_{\mathcal{R}}^*$ .

Lucas and Gutiérrez [19] proposed reducing infeasibility to the model finding of first-order logic. Our formulations especially in Sect. 6 are similar to theirs. A crucial difference is that, while they encode the closure properties and order properties into logical formulas and delegate these tasks to the background theory solvers, we ensure these properties by means of reduction pairs, for which well-established techniques exist in the literature.

Sternagel and Yamada [30] proposed a framework for analyzing reachability by combining basic logical manipulations, and Gutiérrez and Lucas [10] proposed another framework, similar to the dependency pair framework. The present work focuses on atomic analysis techniques, and is orthogonal to these efforts of combining techniques.

## 2 Preliminaries

We assume familiarity with term rewriting, cf. [3] or [32]. For a binary relation denoted by a symbol like  $\sqsupset$ , we denote its dual relation by  $\sqsubset$  and the negated relation by  $\not\sqsupset$ . Relation composition is denoted by  $\circ$ .

Throughout the paper we fix a set  $\mathcal{V}$  of *variable symbols*. A *signature* is a set  $\mathcal{F}$  of function symbols, where each  $f \in \mathcal{F}$  is associated with its *arity*, the number of arguments. The set of *terms* built from  $\mathcal{F}$  and  $\mathcal{V}$  is denoted by  $\mathcal{T}(\mathcal{F}, \mathcal{V})$ , where a term is either in  $\mathcal{V}$  or of form  $f(s_1, \dots, s_n)$  where  $f \in \mathcal{F}$  is  $n$ -ary and  $s_1, \dots, s_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ . Given a term  $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$  and a *substitution*  $\theta : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ ,  $s\theta$  denotes the term obtained from  $s$  by replacing every variable  $x$  by  $\theta(x)$ . A *context* is a term  $C \in \mathcal{T}(\mathcal{F}, \mathcal{V} \cup \{\square\})$  where a special variable  $\square$  occurs exactly once. Given  $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ , we denote by  $C[s]$  the term obtained by replacing  $\square$  by  $s$  in  $C$ .

A relation  $\sqsubset$  over terms is *closed under substitutions (resp. contexts)* iff  $s \sqsubset t$  implies  $s\theta \sqsubset t\theta$  for any substitution  $\theta$  (resp.  $C[s] \sqsubset C[t]$  for any context  $C$ ). Relations over terms that are closed under contexts and substitutions are called *rewrite relations*. Rewrite relations which are also preorders are called *rewrite preorders*, and those which are strict orders are *rewrite orders*. Well-founded rewrite orders are called *reduction orders*.

A *term rewrite system (TRS)*  $\mathcal{R}$  is a (usually finite) relation over terms, where each  $\langle l, r \rangle \in \mathcal{R}$  is called a *rewrite rule* and written  $l \rightarrow r$ . We do not require the usual assumption that  $l \notin \mathcal{V}$  and variables occurring in  $r$  must occur in  $l$ . The *rewrite step*  $\rightarrow_{\mathcal{R}}$  induced by TRS  $\mathcal{R}$  is the least rewrite relation containing  $\mathcal{R}$ . Its reflexive transitive closure is denoted by  $\rightarrow_{\mathcal{R}}^*$ , which is the least rewrite preorder containing  $\mathcal{R}$ .

A *reachability atom* is a pair of terms  $s$  and  $t$ , written  $s \rightarrow t$ . We say that  $s \rightarrow t$  is  $\mathcal{R}$ -*satisfiable* iff  $s\theta \rightarrow_{\mathcal{R}}^* t\theta$  for some  $\theta$ , and  $\mathcal{R}$ -*unsatisfiable* otherwise.

### 3 Term Orderings for Non-reachability

*Reduction pairs* constitute the core ingredient in proving termination with dependency pairs. Just as rewrite orders generalize reduction orders, we first introduce the notion of “rewrite pairs” by removing the well-foundedness assumption of reduction pairs.

**Definition 1 (rewrite pair).** We call a pair  $\langle \sqsupseteq, \sqsubset \rangle$  of relations an *order pair* if  $\sqsupseteq$  is a preorder,  $\sqsubset$  is irreflexive,  $\sqsubset \subseteq \sqsupseteq$ , and  $\sqsupseteq \circ \sqsubset \circ \sqsupseteq \subseteq \sqsubset$ . A *rewrite pair* is an order pair  $\langle \sqsupseteq, \sqsubset \rangle$  over terms such that both  $\sqsupseteq$  and  $\sqsubset$  are closed under substitutions and  $\sqsupseteq$  is closed under contexts. It is called a *reduction pair* if moreover  $\sqsubset$  is well-founded.

Standard definitions of reduction pairs put less order-like assumptions than the above definition, but the above (more natural) assumptions do not lose the generality of previous definitions [34]. Due to these assumptions, our rewrite pair satisfies the assumption of discrimination pairs [1].

The following statement is our first observation: a rewrite pair can prove non-reachability.

**Theorem 1.** If  $\langle \sqsupseteq, \sqsubset \rangle$  is a rewrite pair,  $\mathcal{R} \subseteq \sqsupseteq$  and  $s \sqsubset t$ , then  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable.

A similar observation has been made [20, Theorem 11], where well-foundedness is assumed instead of irreflexivity. Note that irreflexivity is essential: if  $s \sqsubset s$  for some  $s$ , then we have  $s \sqsubset s$  but  $s \rightarrow s$  is  $\mathcal{R}$ -satisfiable.

The proof of Theorem 1 will be postponed until more general Theorem 2 will be obtained. Instead, we start with utilizing Theorem 1 by generalizing a classical way of defining reduction pairs: the semantic approach [23].

**Definition 2 (model).** An  $\mathcal{F}$ -algebra  $\mathcal{A} = \langle A, [\cdot] \rangle$  specifies a set  $A$  called the carrier and an interpretation  $[f] : A^n \rightarrow A$  to each  $n$ -ary  $f \in \mathcal{F}$ . The evaluation of a term  $s$  under assignment  $\alpha : \mathcal{V} \rightarrow A$  is defined as usual and denoted by  $[s]\alpha$ .

A related/preordered  $\mathcal{F}$ -algebra  $\langle \mathcal{A}, \sqsupset \rangle = \langle A, [\cdot], \sqsupset \rangle$  consists of an  $\mathcal{F}$ -algebra and a relation/preorder  $\sqsupset$  on  $A$ . Given  $\alpha : \mathcal{V} \rightarrow A$ , we write  $[s \sqsupset t]\alpha$  to mean  $[s]\alpha \sqsupset [t]\alpha$ . We write  $\mathcal{A} \models s \sqsupset t$  if  $[s \sqsupset t]\alpha$  holds for every  $\alpha : \mathcal{V} \rightarrow A$ . We say  $\langle \mathcal{A}, \sqsupset \rangle$  is a (relational) model of a TRS  $\mathcal{R}$  if  $\mathcal{A} \models l \rightarrow r$  for every  $l \rightarrow r \in \mathcal{R}$ . We say  $\langle \mathcal{A}, \sqsupset \rangle$  is monotone if  $a_i \sqsupset a'_i$  implies  $[f](a_1, \dots, a_i, \dots, a_n) \sqsupset [f](a_1, \dots, a'_i, \dots, a_n)$  for arbitrary  $a_1, \dots, a_n, a'_i \in A$  and  $n$ -ary  $f \in \mathcal{F}$ .

The notion of relational models is due to van Oostrom [28]. In this paper, we simply call them models. Models in terms of equational theory are models  $\langle \mathcal{A}, = \rangle$  in the above definition, where monotonicity is inherent. Quasi-models of Zantema [37] are preordered (or partially ordered) monotone models. Theorem 1 can be reformulated in the semantic manner as follows:

**Corollary 1.** If  $\langle \geq, > \rangle$  is an order pair,  $\langle \mathcal{A}, \geq \rangle$  is a monotone model of  $\mathcal{R}$ , and  $\mathcal{A} \models s < t$ , then  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable.

Note that Corollary 1 does not demand well-foundedness on  $>$ . In particular, one can employ models over negative numbers (or equivalently, positive numbers with the order pair  $\langle \leq, < \rangle$ ).

*Example 2.* Consider again the TRS  $\mathcal{R}_{\text{add}}$  of Example 1. The monotone ordered  $\mathcal{F}$ -algebra  $\langle \mathbb{Z}_{\leq 0}, [\cdot], \geq \rangle$  defined by

$$[\text{add}](x, y) = x + y \quad [\text{s}](x) = x - 1 \quad [0] = 0$$

is a model of  $\mathcal{R}_{\text{add}}$ : Whenever  $x, y \in \mathbb{Z}_{\leq 0}$ , we have

$$[\text{add}]( [0], y ) = y \quad [\text{add}]( [\text{s}](x), y ) = x + y - 1 = [\text{s}]( [\text{add}](x, y) )$$

Now we can conclude that the reachability constraint  $\text{add}(\text{s}(x), y) \rightarrow y$  is  $\mathcal{R}_{\text{add}}$ -unsatisfiable by  $\langle \mathbb{Z}_{\leq 0}, [\cdot] \rangle \models \text{add}(\text{s}(x), y) < y$ : Whenever  $x, y \in \mathbb{Z}_{\leq 0}$ , we have

$$[\text{add}]( [\text{s}](x), y ) = x + y - 1 < y$$

Observe that in Theorem 1,  $\sqsupset$  occurs only in the dual form  $\sqsubset$ . Hence we now directly analyze the condition which  $\sqsupseteq$  and  $\sqsubset$  should satisfy to prove non-reachability, and this gives a sound and complete method.



**Definition 3 (co-rewrite pair).** We call a pair  $\langle \sqsupseteq, \sqsubset \rangle$  of relations over terms a co-rewrite pair, if  $\sqsupseteq$  is a rewrite preorder,  $\sqsubset$  is closed under substitutions, and  $\sqsupseteq \cap \sqsubset = \emptyset$ .

**Theorem 2.**  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable if and only if there exists a co-rewrite pair  $\langle \sqsupseteq, \sqsubset \rangle$  such that  $\mathcal{R} \subseteq \sqsupseteq$  and  $s \sqsubset t$ .

*Proof.* For the “if” direction, suppose on the contrary that  $s\theta \rightarrow_{\mathcal{R}}^* t\theta$  for some  $\theta$ . Since  $\sqsupseteq$  is a rewrite preorder containing  $\mathcal{R}$  and  $\rightarrow_{\mathcal{R}}^*$  is the least of such, we must have  $s\theta \sqsupseteq t\theta$ . On the other hand, since  $s \sqsubset t$  and  $\sqsubset$  is closed under substitutions, we have  $s\theta \sqsubset t\theta$ . This is not possible since  $\sqsupseteq \cup \sqsubset = \emptyset$ .

For the “only if” direction, take  $\rightarrow_{\mathcal{R}}^*$  as  $\sqsupseteq$  and define  $\sqsubset$  by  $s \sqsubset t$  iff  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable. Then clearly  $\sqsubset$  is closed under substitutions,  $\rightarrow_{\mathcal{R}}^* \cap \sqsubset = \emptyset$ , and  $\mathcal{R} \subseteq \rightarrow_{\mathcal{R}}^*$ .  $\square$

Theorem 2 can be more concisely reformulated in the model-oriented manner, as the greatest choice of  $\sqsubset$  can be specified:  $s \sqsubset t$  iff  $\mathcal{A} \models s \not\leq t$ .

**Corollary 2.**  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable if and only if there exists a monotone preordered model  $\langle \mathcal{A}, \geq \rangle$  of  $\mathcal{R}$  such that  $\mathcal{A} \models s \not\leq t$ .

Corollary 2 is useful when models over non-totally ordered carriers are considered. There are important methods (for termination) that crucially rely on such carriers: the *matrix interpretations* [5], or more generally the *tuple interpretations* [15, 34].

*Example 3.* Consider the following TRS, where the first rule is from [5]:

$$\mathcal{R}_{mat} = \{ \mathbf{f}(\mathbf{f}(x)) \rightarrow \mathbf{f}(\mathbf{g}(\mathbf{f}(x))), \mathbf{f}(x) \rightarrow x \}$$

The preordered  $\{\mathbf{f}, \mathbf{g}\}$ -algebra  $\langle \mathbb{N}^2, [\cdot], \geq \rangle$  defined by

$$[\mathbf{f}]\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + y + 1 \\ y + 1 \end{pmatrix} \qquad [\mathbf{g}]\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + 1 \\ 0 \end{pmatrix}$$

is a model of  $\mathcal{R}_{mat}$ , where  $\geq$  is extended pointwise over  $\mathbb{N}^2$ . Indeed, the first rule is oriented as the following calculation demonstrates:

$$[\mathbf{f}]\left([\mathbf{f}]\begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} x + 2y + 3 \\ y + 2 \end{pmatrix} \geq \begin{pmatrix} x + y + 3 \\ 1 \end{pmatrix} = [\mathbf{f}]\left([\mathbf{g}]\left([\mathbf{f}]\begin{pmatrix} x \\ y \end{pmatrix}\right)\right)$$

and the second rule can be easily checked. Now we prove that  $x \rightarrow \mathbf{g}(x)$  is  $\mathcal{R}_{mat}$ -unsatisfiable by Corollary 2. Indeed,  $\langle \mathbb{N}^2, [\cdot] \rangle \models x \not\leq \mathbf{g}(x)$  is shown as follows:

$$\begin{pmatrix} x \\ y \end{pmatrix} \not\geq \begin{pmatrix} x + 1 \\ 0 \end{pmatrix} = [\mathbf{g}]\begin{pmatrix} x \\ y \end{pmatrix}$$

for any  $x, y \in \mathbb{N}$ . Note also that Theorem 1 is not applicable, since  $\langle \mathbb{N}^2, [\cdot] \rangle \not\models x < \mathbf{g}(x)$  due to the second coordinate.

We conclude the section by proving Theorem 1 via Theorem 2.

*Proof (of Theorem 1).* We show that  $\langle \sqsupseteq, \sqsubset \rangle$  form a co-rewrite pair when  $\langle \sqsupseteq, \sqsubset \rangle$  is a rewrite pair. It suffices to show that  $\sqsupseteq \cap \sqsubset = \emptyset$ . To this end, suppose on the contrary that  $s \sqsupseteq t \sqsubset s$ . By compatibility, we have  $s \sqsubset s$ , which contradicts the irreflexivity of  $\sqsubset$ .  $\square$

## 4 Weighted Path Order for Non-reachability

The previous section was concerned with the semantic approach towards obtaining (co-)rewrite pairs. In this section we focus on the syntactic approach. We choose the weighted path order (WPO), which subsumes both the lexicographic path order (LPO) and the Knuth-Bendix order (KBO), so the result of this section applies to these more well-known methods. The *multiset path order* [4] can also be subsumed [29], but we omit this extension to keep the presentation simple. WPO is induced by three ingredients: an  $\mathcal{F}$ -algebra; a *precedence* ordering over function symbols; and a (*partial*) *status*, which controls the recursive behavior of the ordering.

**Definition 4 (partial status).** A partial status  $\pi$  specifies for each  $n$ -ary  $f \in \mathcal{F}$  a list  $\pi(f) \in \{1, \dots, n\}^*$ , also seen as a set, of its argument positions. We say  $\pi$  is *total* if  $1, \dots, n \in \pi(f)$  whenever  $f$  is  $n$ -ary. When  $\pi(f) = [i_1, \dots, i_m]$ , we denote  $[s_{i_1}, \dots, s_{i_m}]$  by  $\pi_f(s_1, \dots, s_n)$ .

For instance, the *empty* status  $\pi(f) = []$  allows WPO to subsume weakly monotone interpretations [36, Section 4.1]. We allow positions to be duplicated, following [33].

**Definition 5 (WPO [36]).** Let  $\pi$  be a partial status,  $\mathcal{A}$  an  $\mathcal{F}$ -algebra, and  $\langle \geq, > \rangle$  and  $\langle \succsim, \succ \rangle$  be pairs of relations on  $\mathcal{A}$  and  $\mathcal{F}$ , respectively. The weighted path order  $\text{WPO}(\pi, \mathcal{A}, \geq, >, \succsim, \succ)$ , or  $\text{WPO}(\mathcal{A})$  or even *WPO* for short, is the pair  $\langle \sqsupseteq_{\text{WPO}}, \sqsubset_{\text{WPO}} \rangle$  of relations over terms defined as follows:  $s \sqsupseteq_{\text{WPO}} t$  iff

1.  $\mathcal{A} \models s > t$  or
2.  $\mathcal{A} \models s \geq t$  and
  - (a)  $s = f(s_1, \dots, s_n)$ ,  $s_i \sqsupseteq_{\text{WPO}} t$  for some  $i \in \pi(f)$ ;
  - (b)  $s = f(s_1, \dots, s_n)$ ,  $t = g(t_1, \dots, t_m)$ ,  $s \sqsupseteq_{\text{WPO}} t_j$  for every  $j \in \pi(g)$  and
    - i.  $f \succ g$ , or
    - ii.  $f \succsim g$  and  $\pi_f(s_1, \dots, s_n) \sqsupset_{\text{WPO}}^{\text{lex}} \pi_g(t_1, \dots, t_m)$ .

The relation  $\sqsupseteq_{\text{WPO}}$  is defined similarly, but with  $\sqsupset_{\text{WPO}}^{\text{lex}}$  instead of  $\sqsupset_{\text{WPO}}^{\text{lex}}$  in (2b-ii) and the following subcase is added in case 2:

- (c)  $s = t \in \mathcal{V}$ .

Here  $\langle \sqsupset_P^{\text{lex}}, \sqsubset_P^{\text{lex}} \rangle$  denotes the lexicographic extension of a pair  $P = \langle \sqsupset_P, \sqsubset_P \rangle$  of relations, defined by:  $[s_1, \dots, s_n] \sqsupset_P^{\text{lex}} [t_1, \dots, t_m]$  iff

- $m = 0$  and  $n \geq 0$ , or
- $m, n > 0$  and  $s_1 > t_1$  or both  $s_1 \sqsupseteq_P t_1$  and  $[s_2, \dots, s_n] \sqsupseteq_P^{\text{lex}} [t_2, \dots, t_m]$ .

LPO is WPO induced by a total status  $\pi$  and a trivial  $\mathcal{F}$ -algebra as  $\mathcal{A}$ , and is written LPO. Allowing partial statuses corresponds to applying *argument filters* [2, 17] (except for collapsing ones). KBO is a special case of WPO where  $\pi$  is total and  $\mathcal{A}$  is induced by an admissible weight function.

For termination analysis, a precondition for WPO to be a reduction pair is crucial. In this work, we only need it to be a rewrite pair; that is, well-foundedness is not necessary. Thus, for instance, it is possible to have  $x \sqsupseteq_{\text{WPO}} f(x)$  by  $[f](x) = x - 1$ . This explains why  $s \in \mathcal{V}$  is permitted in case 1, which might look useless to those who are already familiar with termination analysis.

We formulate the main claim of this section as follows.

**Definition 6 ( $\pi$ -simplicity).** We say a related  $\mathcal{F}$ -algebra  $\langle A, [\cdot], \geq \rangle$  is  $\pi$ -simple<sup>1</sup> for a partial status  $\pi$  iff  $[f](a_1, \dots, a_n) \geq a_i$  for arbitrary  $n$ -ary  $f \in \mathcal{F}$ ,  $a_1, \dots, a_n \in A$ , and  $i \in \pi(f)$ .

**Proposition 1.** If  $\langle \geq, > \rangle$  and  $\langle \succsim, \succ \rangle$  are order pairs on  $\mathcal{A}$  and  $\mathcal{F}$ , and  $\langle \mathcal{A}, \geq \rangle$  is monotone and  $\pi$ -simple, then  $\langle \sqsupseteq_{\text{WPO}}, \sqsupset_{\text{WPO}} \rangle$  is a rewrite pair.

Under these conditions, it is known that  $\sqsupseteq_{\text{WPO}}$  is closed under contexts and  $\sqsupset_{\text{WPO}}$  is compatible with  $\sqsupseteq_{\text{WPO}}$  [36, Lemmas 7, 10, 13]. Later in this section we prove other properties necessary for Proposition 1, for which the claims in [36] must be generalized for the purpose of this paper.

The benefit of having syntax-aware methods can be easily observed by recalling why we have them in termination analysis.

*Example 4* ([13]). Consider the TRS  $\mathcal{R}_{\mathcal{A}}$  consisting of the following rules:

$$\mathbf{A}(0, y) \rightarrow \mathbf{s}(y) \quad \mathbf{A}(\mathbf{s}(x), 0) \rightarrow \mathbf{A}(x, \mathbf{s}(0)) \quad \mathbf{A}(\mathbf{s}(x), \mathbf{s}(y)) \rightarrow \mathbf{A}(x, \mathbf{A}(\mathbf{s}(x), y))$$

and suppose that a monotone  $\{\mathbf{A}, \mathbf{s}, 0\}$ -algebra  $\langle \mathbb{N}, [\cdot], \geq \rangle$  is a model of  $\mathcal{R}_{\mathcal{A}}$ . Then, denoting the Ackermann function by  $A$ , we have

$$[\mathbf{A}](\mathbf{s}^m(0), \mathbf{s}^n(0)) \geq \mathbf{s}^{A(m,n)}(0) \quad (1)$$

Now consider proving the obvious fact that  $x \rightarrow \mathbf{s}(x)$  is  $\mathcal{R}_{\mathcal{A}}$ -unsatisfiable. This requires  $\langle \mathbb{N}, [\cdot] \rangle \models x < \mathbf{s}(x)$ , and then  $\mathbf{s}^n(0) \geq n$  by an inductive argument. This is not possible if  $[\mathbf{A}]$  is primitive recursive (e.g., a polynomial), since (1) with  $\mathbf{s}^{A(m,n)}(0) \geq A(m, n)$  contradicts the well-known fact that the Ackermann function has no primitive-recursive bound.

On the other hand, LPO with  $\mathbf{A} \succ \mathbf{s}$  satisfies  $\mathcal{R}_{\mathcal{A}} \subseteq \sqsupseteq_{\text{LPO}} (\subseteq \sqsupseteq_{\text{LPO}})$  and  $x \sqsubset_{\text{LPO}} \mathbf{s}(x)$ . Thus Theorem 1 with  $\langle \sqsupseteq, \sqsupset \rangle = \langle \sqsupseteq_{\text{LPO}}, \sqsupset_{\text{LPO}} \rangle$  proves that  $x \rightarrow \mathbf{s}(x)$  is  $\mathcal{R}_{\mathcal{A}}$ -unsatisfiable, thanks to Proposition 1 and Theorem 1.

<sup>1</sup> Such a property would be called *inflationary* in the mathematics literature. In the term rewriting, the word *simple* has been used (see, e.g., [32]) in accordance with *simplification orders*.

*Example 5.* Consider the TRS consisting of the following rules:

$$\mathcal{R}_{kbo} := \{ \mathbf{f}(\mathbf{g}(x)) \rightarrow \mathbf{g}(\mathbf{f}(\mathbf{f}(x))), \mathbf{g}(x) \rightarrow x \}$$

WPO (or KBO) induced by  $\mathcal{A} = \langle \mathbb{N}, [\cdot] \rangle$  and precedence  $\langle \succsim, \succ \rangle$  such that

$$[\mathbf{f}](x) = x \qquad [\mathbf{g}](x) = x + 1 \qquad \mathbf{f} \succ \mathbf{g}$$

satisfies  $\mathcal{R}_{kbo} \subseteq \sqsubseteq_{\text{WPO}}$ . Thus, for instance  $\mathbf{g}(x) \rightarrow \mathbf{g}(\mathbf{f}(x))$  is  $\mathcal{R}_{kbo}$ -unsatisfiable by Theorem 1. On the other hand, let  $\langle A, [\cdot], \geq \rangle$  with  $A \subseteq \mathbb{Z}$  be a model of  $\mathcal{R}_{kbo}$ . Using the idea of [38, Proposition 11], one can show  $[\mathbf{f}](x) \leq x$ . Hence, Corollary 2 with models over a subset of integers cannot handle the problem. LPO orients the first rule from right to left and hence cannot handle the problem either.

The power of WPO can also be easily verified, by considering

$$\mathcal{R}_{wpo} := \mathcal{R}_{kbo} \cup \{ \mathbf{f}(\mathbf{h}(x)) \rightarrow \mathbf{h}(\mathbf{f}(x)), \mathbf{f}(x) \rightarrow x \}$$

By extending the above WPO with  $[\mathbf{h}](x) = x$  and  $\mathbf{f} \succ \mathbf{h}$ , which does not fall into the class of KBO anymore,<sup>2</sup> we can prove, e.g., that  $\mathbf{f}(x) \rightarrow \mathbf{f}(\mathbf{h}(x))$  is  $\mathcal{R}$ -unsatisfiable. None of the above mentioned methods can handle this problem.

The rest of this section is dedicated for proving Proposition 1. Similar results are present in [36], but they make implicit assumptions such as that  $\geq$  and  $\succsim$  are preorders. In this paper we need more essential assumptions as we will consider non-transitive relations in the next section.

First we reprove the reflexivity of  $\sqsubseteq_{\text{WPO}}$ . The proof also serves as a basis for the more complicated irreflexivity proof.

**Lemma 1.** *If both  $\geq$  and  $\succsim$  are reflexive and  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple, then*

1.  *$i \in \pi(f)$  implies  $f(s_1, \dots, s_n) \sqsubseteq_{\text{WPO}} s_i$ , and*
2.  *$s \sqsubseteq_{\text{WPO}} s$ , i.e.,  $\sqsubseteq_{\text{WPO}}$  is reflexive.*

*Proof.* As  $s \sqsubseteq_{\text{WPO}} s$  is trivial when  $s \in \mathcal{V}$ , we assume  $s = f(s_1, \dots, s_n)$  and prove the two claims by induction on the structure of  $s$ . For the first claim, by  $\pi$ -simplicity, for any  $\alpha$  we have  $[s]\alpha = [f]([s_1]\alpha, \dots, [s_n]\alpha) \geq [s_i]\alpha$ , and hence  $\mathcal{A} \models s \geq s_i$ . By the second claim of induction hypothesis we have  $s_i \sqsubseteq_{\text{WPO}} s_i$ , and thus  $s \sqsubseteq_{\text{WPO}} s_i$  follows by (2a) of Definition 5. Next we show  $s \sqsubseteq_{\text{WPO}} s$  holds by (2b-ii). Indeed,  $\mathcal{A} \models s \geq s$  follows from the reflexivity of  $\geq$ ;  $s \sqsubseteq_{\text{WPO}} s_i$  for every  $i \in \pi(f)$  as shown above;  $f \succsim f$  as  $\succsim$  is reflexive; and finally,  $\pi_f(s_1, \dots, s_n) \sqsubseteq_{\text{WPO}}^{\text{lex}} \pi_f(s_1, \dots, s_n)$  is due to induction hypothesis and the fact that lexicographic extension preserves reflexivity.  $\square$

Using reflexivity, we can show that both  $\sqsubseteq_{\text{WPO}}$  and  $\sqsubset_{\text{WPO}}$  are closed under substitutions. This result will be reused in Sect. 5, where it will be essential that neither  $\geq$  nor  $>$  need be transitive.

<sup>2</sup> When  $[\mathbf{h}]$  is the identity. KBO requires  $\mathbf{h} \succsim f$  for any  $f$ .

**Lemma 2.** *If both  $\geq$  and  $\succsim$  are reflexive and  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple, then both  $\sqsubseteq_{\text{WPO}}$  and  $\sqsupset_{\text{WPO}}$  are closed under substitutions.*

*Proof.* We prove by induction on  $s$  and  $t$  that  $s \sqsubseteq_{\text{WPO}} t$  implies  $s\theta \sqsubseteq_{\text{WPO}} t\theta$  and that  $s \sqsupset_{\text{WPO}} t$  implies  $s\theta \sqsupset_{\text{WPO}} t\theta$ . We prove the first claim by case analysis on how  $s \sqsubseteq_{\text{WPO}} t$  is derived. The other claim is analogous, without case (2c) below.

1.  $\mathcal{A} \models s > t$ : Then we have  $\mathcal{A} \models s\theta > t\theta$  and thus  $s\theta \sqsupset_{\text{WPO}} t\theta$  by case 1.
2.  $\mathcal{A} \models s \geq t$ : Then we have  $\mathcal{A} \models s\theta \geq t\theta$ . There are the following subcases.
  - (a)  $s = f(s_1, \dots, s_n)$  and  $s_i \sqsubseteq_{\text{WPO}} t$  for some  $i \in \pi(f)$ : In this case, we know  $s_i\theta \sqsubseteq_{\text{WPO}} t\theta$  by induction hypothesis on  $s$ . Thus (2a) concludes  $s\theta \sqsubseteq_{\text{WPO}} t\theta$ .
  - (b)  $s = f(s_1, \dots, s_n)$ ,  $t = g(t_1, \dots, t_m)$ , and  $s \sqsupset_{\text{WPO}} t_j$  for every  $j \in \pi(g)$ : By induction hypothesis on  $t$ , we have  $s\theta \sqsupset_{\text{WPO}} t_j\theta$ . So the precondition of (2b) for  $s\theta \sqsubseteq_{\text{WPO}} t\theta$  is satisfied. There are the following subcases:
    - i.  $f \succ g$ : Then (2b-i) concludes.
    - ii.  $f \succsim g$  and  $\pi_f(s_1, \dots, s_n) \sqsubseteq_{\text{WPO}}^{\text{lex}} \pi_g(t_1, \dots, t_m)$ : Then by induction hypothesis we have  $\pi_f(s_1\theta, \dots, s_n\theta) \sqsubseteq_{\text{WPO}}^{\text{lex}} \pi_g(t_1\theta, \dots, t_m\theta)$ , and thus (2b-ii) concludes.
  - (c)  $s = t \in \mathcal{V}$ : Then we have  $s\theta \sqsubseteq_{\text{WPO}} t\theta$  by Lemma 1. □

Irreflexivity of  $\sqsubseteq_{\text{WPO}}$  is less obvious to have. In fact, [36] uses well-foundedness to claim it. Here we identify more essential conditions.

**Lemma 3.** *If  $\langle \geq, > \rangle$  is an order pair on  $\mathcal{A}$ , and  $\succ$  is irreflexive on  $\mathcal{F}$ , and  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple, then  $\sqsubseteq_{\text{WPO}}$  is irreflexive.*

*Proof.* We show  $s \not\sqsubseteq_{\text{WPO}} s$  for every  $s$  by induction on the structure of  $s$ . This is clear if  $s \in \mathcal{V}$ , so consider  $s = f(s_1, \dots, s_n)$ . Since  $>$  is irreflexive, we have  $\mathcal{A} \not\models s > s$ , and thus  $s \sqsubseteq_{\text{WPO}} s$  cannot be due to case 1 of Definition 5. As  $\succ$  is irreflexive on  $\mathcal{F}$ ,  $f \not\succ f$  and thus (2b-i) is not possible, either. Thanks to induction hypothesis and the fact that lexicographic extension preserves irreflexivity, we have  $\pi_f(s_1, \dots, s_n) \not\sqsubseteq_{\text{WPO}}^{\text{lex}} \pi_f(s_1, \dots, s_n)$ , and thus (2b-ii) is not possible either.

The remaining (2a) is more involving. To show  $s_i \not\sqsubseteq_{\text{WPO}} f(s_1, \dots, s_n)$  for any  $i \in \pi(f)$ , we prove the following more general claim:  $s' \triangleleft_{\pi}^+ s$  implies  $s' \not\sqsubseteq_{\text{WPO}} s$ , where  $\triangleleft_{\pi}$  denotes the least relation such that  $s_i \triangleleft_{\pi} f(s_1, \dots, s_n)$  if  $i \in \pi(f)$ . This claim is proved by induction on  $s'$ . Due to the simplicity assumption, we have  $\mathcal{A} \models s \geq s'$  for every  $s' \triangleleft_{\pi} s$ , and this generalizes for every  $s' \triangleleft_{\pi}^+ s$  by easy induction and the transitivity of  $\geq$ . Thus we cannot have  $\mathcal{A} \models s' > s$ , since  $\mathcal{A} \models s \geq s' > s$  contradicts the assumption that  $\langle \geq, > \rangle$  is an order pair. This tells us that  $s' \sqsubseteq_{\text{WPO}} s$  cannot be due to case 1. Case (2a) is not applicable thanks to (inner) induction hypothesis on  $s'$ . Case (2b) is not possible either, since  $s' \not\sqsubseteq_{\text{WPO}} s'$  thanks to (outer) induction hypothesis on  $s$ . This concludes  $s' \not\sqsubseteq_{\text{WPO}} s$  for any  $s' \triangleleft_{\pi}^+ s$ , and in particular  $s_i \not\sqsubseteq_{\text{WPO}} s$  for any  $i \in \pi(f)$ , refuting the last possibility for  $s \sqsubseteq_{\text{WPO}} s$  to hold. □

## 5 Co-WPO

The preceding section demonstrated how to use WPO as a rewrite pair in Theorem 1. In this section we show how to use WPO in combination with Theorem 2, that is, when  $\sqsupseteq = \sqsupseteq_{\text{WPO}}$ , what  $\sqsubset$  should be. We show that  $\sqsubset_{\overline{\text{WPO}}}$ , where  $\overline{\text{WPO}} := \text{WPO}(\pi, \mathcal{A}, \not\leq, \not\leq, \not\leq, \not\leq)$ , serves the purpose.

**Proposition 2.** *If  $\langle \geq, > \rangle$  and  $\langle \succeq, \succ \rangle$  are order pairs on  $\mathcal{A}$  and  $\mathcal{F}$ ,  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple and monotone, then  $\langle \sqsupseteq_{\text{WPO}}, \sqsubset_{\overline{\text{WPO}}} \rangle$  is a co-rewrite pair.*

When  $\langle \mathcal{A}, \geq \rangle$  is not total, Example 3 also demonstrates that using Proposition 2 with Theorem 2 is more powerful than using Proposition 1 in combination with Theorem 1, by taking  $\pi(f) = []$  for every  $f$ . At the time of writing, however, it is unclear to the author if the difference still exists when  $\langle \mathcal{A}, \geq \rangle$  is totally ordered but  $\langle \mathcal{F}, \succeq \rangle$  is not. Nevertheless we will clearly see the merit of Proposition 2 under the setting of conditional rewriting in the next section.

The remainder of this section proves Proposition 2. Unfortunately,  $\overline{\text{WPO}}$  does not satisfy many important properties of WPO, mostly due to the fact that  $\langle \not\leq, \not\leq \rangle$  is not even an order pair. Nevertheless, Lemma 2 is applicable to  $\sqsubset_{\overline{\text{WPO}}}$  and gives the following fact:

**Lemma 4.** *If  $\langle \geq, > \rangle$  is an order pair on  $\mathcal{A}$ ,  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple, and  $\succ$  is irreflexive, then  $\sqsubset_{\overline{\text{WPO}}}$  is closed under substitutions.*

*Proof.* We apply Lemma 2 to  $\overline{\text{WPO}}$ . To this end, we need to prove the following:

- $\langle \mathcal{A}, \not\leq \rangle$  is  $\pi$ -simple: Suppose on the contrary one had  $[f](a_1, \dots, a_n) < a_i$  with  $i \in \pi(f)$ . Due to the simplicity assumption, we have  $[f](a_1, \dots, a_n) \geq a_i$ . By compatibility we must have  $a_i < a_i$ , contradicting irreflexivity.
- $\not\leq$  and  $\not\leq$  are reflexive: This follows from the irreflexivity of  $<$  and  $\prec$ .  $\square$

The remaining task is to show that  $\sqsupseteq_{\text{WPO}} \cap \sqsubset_{\overline{\text{WPO}}} = \emptyset$ . Due to the mutual inductive definition of WPO, we need to simultaneously prove the property for the other combination:  $\sqsupseteq_{\overline{\text{WPO}}} \cap \sqsubset_{\text{WPO}} = \emptyset$ .

**Definition 7.** *We say that two pairs  $P = \langle \sqsupseteq_P, \sqsubset_P \rangle$  and  $Q = \langle \sqsupseteq_Q, \sqsubset_Q \rangle$  of relations are co-compatible iff  $\sqsupseteq_P \cap \sqsubset_Q = \sqsubset_P \cap \sqsupseteq_Q = \emptyset$ .*

The next claim is a justification for the word “compatible” in Definition 7. Here the compatibility assumption of order pairs is crucial.

**Proposition 3.** *An order pair  $\langle \sqsupseteq, \sqsubset \rangle$  is co-compatible with itself.*

*Proof.* Suppose on the contrary that  $a \sqsupseteq b$  and  $b \sqsubset a$ . Then we have  $a \sqsubset a$  by compatibility, contradicting the irreflexivity of  $\sqsubset$ .  $\square$

**Lemma 5.** *If  $P = \langle \sqsupseteq_P, \sqsubset_P \rangle$  and  $Q = \langle \sqsupseteq_Q, \sqsubset_Q \rangle$  are co-compatible pairs of relations, then  $\langle \sqsupseteq_P^{\text{lex}}, \sqsubset_P^{\text{lex}} \rangle$  and  $\langle \sqsupseteq_Q^{\text{lex}}, \sqsubset_Q^{\text{lex}} \rangle$  are co-compatible.*

*Proof.* Let us assume that both

$$[s_1, \dots, s_n] \supseteq_P^{\text{lex}} [t_1, \dots, t_m] \quad (2)$$

$$[s_1, \dots, s_n] \sqsubset_Q^{\text{lex}} [t_1, \dots, t_m] \quad (3)$$

hold and derive a contradiction. The other part  $\supseteq_P^{\text{lex}} \cap \sqsubseteq_Q^{\text{lex}}$  is analogous. We proceed by induction on the length of  $[s_1, \dots, s_n]$ . If  $n = 0$ , then (2) demands  $m = 0$  but (3) demands  $m > 0$ . Hence we have  $n > 0$ , and then (3) demands  $m > 0$ . If  $s_1 \supseteq_P t_1$  then by assumption we have  $s_1 \not\sqsubseteq_Q t_1$  but (3) demands  $s_1 \sqsubseteq_Q t_1$  (or  $s_1 \sqsubset_Q t_1$ ). Hence (2) is due to  $s_1 \supseteq_P t_1$  and  $[s_2, \dots, s_n] \supseteq_P^{\text{lex}} [t_2, \dots, t_m]$ . By assumption we have  $s_1 \not\sqsubseteq_Q t_1$ , so (3) is due to  $s_1 \sqsubseteq_Q t_1$  and  $[s_2, \dots, s_n] \sqsubset_Q^{\text{lex}} [t_2, \dots, t_m]$ . We derive a contradiction by induction hypothesis.  $\square$

We arrive at the main lemma for  $\overline{\text{WPO}}$ .

**Lemma 6.** *If  $\langle \geq, > \rangle$  and  $\langle \succsim, \succ \rangle$  are order pairs on  $\mathcal{A}$  and  $\mathcal{F}$ , and  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple, then  $\text{WPO}$  and  $\overline{\text{WPO}}$  are co-compatible.*

*Proof.* We show that neither  $s \supseteq_{\text{WPO}} t \wedge s \sqsubset_{\overline{\text{WPO}}} t$  nor  $s \supset_{\text{WPO}} t \wedge s \sqsubseteq_{\overline{\text{WPO}}} t$  hold for any  $s$  and  $t$ , by induction on the structure of  $s$  and then  $t$ . Let us assume  $s \supseteq_{\text{WPO}} t$  and prove  $s \not\sqsubset_{\overline{\text{WPO}}} t$ . The other claim is analogous. We proceed by case analysis on the derivation of  $s \supseteq_{\text{WPO}} t$ .

1.  $\mathcal{A} \models s > t$ : Then  $s \sqsubset_{\overline{\text{WPO}}} t$  cannot hold as it demands  $\mathcal{A} \models s \not\geq t$  (or  $s \not\geq t$ ).
2.  $\mathcal{A} \models s \geq t$ : Then  $\mathcal{A} \models s \not\geq t$  cannot happen and thus  $s \sqsubset_{\overline{\text{WPO}}} t$  must be due to case 2 of Definition 5. There are the following subcases for  $s \supseteq_{\text{WPO}} t$ :
  - (a)  $s = f(s_1, \dots, s_n)$ ,  $s_i \supseteq_{\text{WPO}} t$  for some  $i \in \pi(f)$ : By induction hypothesis on  $s$ , we have  $s_i \not\sqsubset_{\overline{\text{WPO}}} t$ , and thus  $s \sqsubset_{\overline{\text{WPO}}} t$  can only be due to (2a). So  $t = g(t_1, \dots, t_m)$  and  $s \sqsubseteq_{\overline{\text{WPO}}} t_j$  for some  $j \in \pi(g)$ . Then  $s \not\sqsupset_{\text{WPO}} t_j$  by induction hypothesis on  $t$ . On the contrary we must have  $s \supset_{\text{WPO}} t_j$ : By Lemma 1–1. we have  $s \supset_{\text{WPO}} s_i \supseteq_{\text{WPO}} t \supset_{\text{WPO}} t_j$  and hence  $s \supset_{\text{WPO}} t_j$  as  $\langle \supset_{\text{WPO}}, \supset_{\text{WPO}} \rangle$  is an order pair.
  - (b)  $s = f(s_1, \dots, s_n)$ ,  $t = g(t_1, \dots, t_m)$ , and  $s \supset_{\text{WPO}} t_j$  for every  $j \in \pi(g)$ : By induction hypothesis on  $t$ , we have  $s \not\sqsubseteq_{\overline{\text{WPO}}} t_j$  for any  $j \in \pi(g)$ . Thus  $s \sqsubset_{\overline{\text{WPO}}} t$  must be due to (2b). We proceed by further considering the following two possibilities.
    - i.  $f \succ g$ : As neither  $f \not\geq g$  nor  $f \not\geq g$  hold,  $s \sqsubseteq_{\overline{\text{WPO}}} t$  is not possible.
    - ii.  $f \succsim g$  and  $\pi_f(s_1, \dots, s_n) \supseteq_{\text{WPO}}^{\text{lex}} \pi_g(t_1, \dots, t_m)$ : As  $f \not\geq g$  does not hold, (2b-i) is not applicable to have  $s \sqsubset_{\overline{\text{WPO}}} t$ . By Lemma 5 and induction hypothesis, we have  $\pi_f(s_1, \dots, s_n) \not\sqsubseteq_{\text{WPO}}^{\text{lex}} \pi_g(t_1, \dots, t_m)$  and thus (2b-ii) is also not applicable, either.
  - (c)  $s = t \in \mathcal{V}$ : Then clearly  $s \sqsubset_{\overline{\text{WPO}}} t$  cannot hold.  $\square$

## 6 Conditional Rewriting

Conditional term rewriting (cf. [27]) is an extension of term rewriting so that rewrite rules can be guarded by conditions. We are interested in the “oriented” variants, as they naturally correspond to functional programming concepts such as **where** clauses of Haskell or **when** clauses of OCaml.

A *conditional rewrite rule*  $l \rightarrow r \Leftarrow \phi$  consists of terms  $l$  and  $r$ , and a list  $\phi$  of pairs of terms. We may omit “ $\Leftarrow []$ ” and write  $s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n$  for  $[(s_1, t_1), \dots, (s_n, t_n)]$ . A *conditional TRS (CTRS)*  $\mathcal{R}$  is a set of conditional rewrite rules. A CTRS  $\mathcal{R}$  yields the rewrite preorder  $\rightarrow_{\mathcal{R}}^*$  by the following derivation rules [22]:

$$\begin{array}{c} \frac{}{s \rightarrow_{\mathcal{R}}^* s} \text{REFL} \quad \frac{s \rightarrow_{\mathcal{R}} t \quad t \rightarrow_{\mathcal{R}}^* u}{s \rightarrow_{\mathcal{R}}^* u} \text{TRANS} \\[10pt] \frac{s_i \rightarrow_{\mathcal{R}} s'_i}{f(s_1, \dots, s_i, \dots, s_n) \rightarrow_{\mathcal{R}} f(s_1, \dots, s'_i, \dots, s_n)} \text{MONO} \\[10pt] \frac{s_1 \theta \rightarrow_{\mathcal{R}}^* t_1 \theta \quad \dots \quad s_n \theta \rightarrow_{\mathcal{R}}^* t_n \theta}{l \theta \rightarrow_{\mathcal{R}} r \theta} \text{RULE} \quad \text{if } (l \rightarrow r \Leftarrow s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n) \in \mathcal{R} \end{array}$$

To approximate reachability with respect to CTRSs by means of (co-)rewrite pairs, one needs to be careful when dealing with conditions.

*Example 6.* Consider the following CTRS:

$$\mathcal{R}_{\text{fg}} := \{ \textcolor{blue}{f}(x) \rightarrow \textcolor{red}{x}, \textcolor{green}{g}(x) \rightarrow \textcolor{blue}{y} \Leftarrow \textcolor{blue}{f}(x) \rightarrow \textcolor{blue}{y} \}$$

and a reachability atom  $\textcolor{green}{g}(x) \rightarrow \textcolor{blue}{f}(x)$ . One might expect that a rewrite preorder  $\sqsupseteq$  such that

$$\textcolor{blue}{f}(x) \sqsupseteq \textcolor{red}{x} \qquad \textcolor{green}{g}(x) \sqsupseteq \textcolor{blue}{y} \text{ if } \textcolor{blue}{f}(x) \sqsupseteq \textcolor{blue}{y}$$

can over-approximate  $\rightarrow_{\mathcal{R}_{\text{fg}}}^*$ , but this is unfortunately false. For instance, any LPO satisfies the above constraints:  $\textcolor{blue}{f}(x) \sqsupseteq_{\text{LPO}} \textcolor{red}{x}$  as LPO is a simplification order, and the second constraints also vacuously holds as the condition  $\textcolor{blue}{f}(x) \sqsupseteq_{\text{LPO}} \textcolor{blue}{y}$  is false. However, it is unsound to conclude that  $\textcolor{green}{g}(x) \rightarrow \textcolor{blue}{f}(x)$  is  $\mathcal{R}_{\text{fg}}$ -unsatisfiable even if  $\textcolor{green}{g}(x) \sqsubset_{\text{LPO}} \textcolor{blue}{f}(x)$ : by setting  $\textcolor{blue}{f} \succ \textcolor{green}{g}$  one can have  $\textcolor{green}{g}(x) \sqsubset_{\text{LPO}} \textcolor{blue}{f}(x)$  and  $\textcolor{green}{g}(x) \sqsubset_{\text{LPO}} \textcolor{blue}{f}(x)$ , but  $\textcolor{green}{g}(x) \rightarrow_{\mathcal{R}_{\text{fg}}} \textcolor{blue}{f}(x)$ .

A solution is to use co-rewrite pairs already for dealing with conditions.

**Proposition 4.** *If  $\langle \sqsupseteq, \sqsubset \rangle$  is a co-rewrite pair,  $(l \rightarrow r \Leftarrow \phi) \in \mathcal{R}$  implies  $l \sqsupseteq r$  or  $u \sqsubset v$  for some  $u \rightarrow v \in \phi$ , and  $s \sqsubset t$ , then  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable.*

*Proof.* We show that  $s \rightarrow_{\mathcal{R}}^* t$  implies  $s \sqsupseteq t$ . This is sufficient, since, then  $s \theta \rightarrow_{\mathcal{R}}^* t \theta$  implies  $s \theta \sqsupseteq t \theta$ , while  $s \sqsubset t$  demands  $s \theta \sqsubset t \theta$ , which is not possible since  $\sqsupseteq \cap \sqsubset = \emptyset$ . The claim is proved by induction on the derivation of  $s \rightarrow_{\mathcal{R}}^* t$ .

– REFL: Since  $\sqsupseteq$  is reflexive, we have  $s \sqsupseteq s$ .



- TRANS: We have  $s \rightarrow_{\mathcal{R}} t$  and  $t \rightarrow_{\mathcal{R}}^* u$  as premises, and  $s \sqsupseteq t$  and  $t \sqsupseteq u$  by induction hypothesis. Since  $\sqsupseteq$  is transitive we conclude  $s \sqsupseteq u$ .
- MONO: We have  $s_i \rightarrow_{\mathcal{R}} s'_i$  as a premise and  $s_i \sqsupseteq s'_i$  by induction hypothesis. Since  $\sqsupseteq$  is closed under contexts, we get  $f(s_1, \dots, s_i, \dots, s_n) \sqsupseteq f(s_1, \dots, s'_i, \dots, s_n)$ .
- RULE: We have  $(l \rightarrow r \Leftarrow s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n) \in \mathcal{R}$ , and for every  $i \in \{1, \dots, n\}$  have  $s_i \theta \rightarrow_{\mathcal{R}}^* t_i \theta$  as a premise and  $s_i \theta \sqsupseteq t_i \theta$  by induction hypothesis. Since  $\sqsupseteq \cap \sqsubset = \emptyset$ , we get  $s_i \theta \not\sqsubset t_i \theta$ . Since  $\sqsubset$  is closed under substitutions, we conclude  $s_i \not\sqsubset t_i$  for every  $i \in \{1, \dots, n\}$ . By assumption, this entails  $l \sqsupseteq r$ , and since  $\sqsupseteq$  is closed under substitution, we conclude  $l\theta \sqsupseteq r\theta$ .  $\square$

*Example 7.* Consider the following singleton CTRS:

$$\mathcal{R}_{ab} := \{ a \rightarrow b \Leftarrow b \rightarrow a \}$$

Proposition 4 combined with LPO or WPO induced by a partial precedence such that  $a \not\prec b$  and  $b \not\prec a$  proves that  $a \rightarrow b$  is  $\mathcal{R}_{ab}$ -unsatisfiable: Clearly  $b \sqsubseteq_{\text{LPO}} a$  and  $a \sqsubseteq_{\text{LPO}} b$  by case (2b-i) of Definition 5. On the other hand, Proposition 4 with the term ordering induced by a totally ordered algebra  $\langle \mathcal{A}, \geq \rangle$  cannot solve the problem, since  $\mathcal{A} \models a \not\prec b$  implies  $\mathcal{A} \models b \geq a$  by totality, which then demands  $\mathcal{A} \models a \geq b$  to satisfy the assumption of Proposition 4. For the same reason, WPO induced by a totally ordered algebra and a total precedence cannot handle the problem either.

Note that the condition of the rule in  $\mathcal{R}_{ab}$  is unsatisfiable, and this is one of the two cases where Proposition 4 is effective. The other case is when a condition can be ignored. Proposition 4 is incomplete when conditions are essential, as in Example 6. For dealing with essential conditional rules, the variable binding in a rule should be taken into account. At this point, a model-oriented formulation (*a la* [19]) seems more suitable.

**Definition 8 (model of CTRS).** We extend the notation  $[s \sqsubset t]\alpha$  of Definition 2 to  $[\phi]\alpha$  for an arbitrary Boolean formula  $\phi$  with the single binary predicate  $\sqsubset$  in the obvious manner. We say  $\mathcal{A} = \langle \mathcal{A}, [\cdot] \rangle$  validates  $\phi$ , written  $\mathcal{A} \models \phi$ , iff  $[\phi]\alpha$  for every  $\alpha : \mathcal{V} \rightarrow \mathcal{A}$ . We say a related  $\mathcal{F}$ -algebra  $\langle \mathcal{A}, \sqsubset \rangle$  is a model of a CTRS  $\mathcal{R}$  iff<sup>3</sup>  $\mathcal{A} \models l \sqsubset r \vee s_1 \not\sqsubset t_1 \vee \dots \vee s_n \not\sqsubset t_n$  for every  $(l \rightarrow r \Leftarrow s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n) \in \mathcal{R}$ .

Besides minor simplifications (e.g., we do not need two predicates as we are only concerned with reachability in many steps in this paper), the major difference with [19] is that here we do not encode the monotonicity or order axioms into logical formulas (using  $\overline{\mathcal{R}}$  of [19]). Instead, we impose these properties as meta-level assumptions over models.

**Theorem 3.** For a CTRS  $\mathcal{R}$ ,  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable if and only if there exists a monotone preordered model  $\langle \mathcal{A}, \geq \rangle$  of  $\mathcal{R}$  such that  $\mathcal{A} \models s \not\prec t$ .

<sup>3</sup> Here the formula  $s \not\prec t$  is a shorthand for  $\neg s \sqsubset t$ .

*Proof.* We start with the “if” direction. Let  $\langle \mathcal{A}, \geq \rangle$  be a monotone preordered model of  $\mathcal{R}$ . As in Proposition 4, it suffices to show that  $s \rightarrow_{\mathcal{R}}^* t$  implies  $\mathcal{A} \models s \geq t$ . The claim is proved by induction on the derivation of  $s \rightarrow_{\mathcal{R}}^* t$ .

- REF: Since  $\geq$  is reflexive, we have  $\mathcal{A} \models s \geq s$ .
- TRANS: We have  $s \rightarrow_{\mathcal{R}} t$  and  $t \rightarrow_{\mathcal{R}}^* u$  as premises, and  $\mathcal{A} \models s \geq t$  and  $\mathcal{A} \models t \geq u$  by induction hypothesis. Since  $\geq$  is transitive we conclude  $\mathcal{A} \models s \geq u$ .
- MONO: We have  $s_i \rightarrow_{\mathcal{R}} s'_i$  as a premise and  $\mathcal{A} \models s_i \geq s'_i$  by induction hypothesis. Since  $\langle \mathcal{A}, \geq \rangle$  is monotone, we get  $\mathcal{A} \models f(s_1, \dots, s_i, \dots, s_n) \geq f(s_1, \dots, s'_i, \dots, s_n)$ .
- RULE: We have  $(l \rightarrow r \Leftarrow s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n) \in \mathcal{R}$ , and for every  $i \in \{1, \dots, n\}$  have  $s_i \theta \rightarrow_{\mathcal{R}}^* t_i \theta$  as a premise and  $\mathcal{A} \models s_i \theta \geq t_i \theta$  by induction hypothesis. Since  $\langle \mathcal{A}, \geq \rangle$  is a model of  $\mathcal{R}$ , and by the fact that validity is closed under substitutions, we have  $\mathcal{A} \models l \theta \geq r \theta \vee s_1 \theta \not\geq t_1 \theta \vee \dots \vee s_n \theta \not\geq t_n \theta$ . Together with the induction hypotheses we conclude  $\mathcal{A} \models l \theta \geq r \theta$ .

Next consider the “only if” direction. We show that  $\langle \mathcal{T}(\mathcal{F}, \mathcal{V}), \rightarrow_{\mathcal{R}}^* \rangle$  is a model of  $\mathcal{R}$ , that is, for every  $(l \rightarrow r \Leftarrow s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n) \in \mathcal{R}$ , we show  $\mathcal{T}(\mathcal{F}, \mathcal{V}) \models l \rightarrow_{\mathcal{R}}^* r \vee s_1 \not\rightarrow_{\mathcal{R}}^* t_1 \vee \dots \vee s_n \not\rightarrow_{\mathcal{R}}^* t_n$ . This means  $l \theta \rightarrow_{\mathcal{R}}^* r \theta$  for every  $\theta : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$  such that  $s_1 \theta \rightarrow_{\mathcal{R}}^* t_1 \theta, \dots, s_n \theta \rightarrow_{\mathcal{R}}^* t_n \theta$ , which is immediate by RULE. The fact that  $\rightarrow_{\mathcal{R}}^*$  is a preorder and closed under contexts is also immediate. Finally,  $s \rightarrow t$  being  $\mathcal{R}$ -unsatisfiable means that  $s \theta \not\rightarrow_{\mathcal{R}}^* t \theta$  for any  $\theta : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ , that is,  $\mathcal{T}(\mathcal{F}, \mathcal{V}) \models s \not\rightarrow_{\mathcal{R}}^* t$ .  $\square$

Putting implementation issues aside, it is trivial to use semantic (termination) methods in Theorem 3.

*Example 8.* Consider again the CTRS  $\mathcal{R}_{fg}$  of Example 6. The monotone ordered  $\{\mathbf{f}, \mathbf{g}\}$ -algebra  $\langle \mathbb{N}, [\cdot], \geq \rangle$  defined by

$$[\mathbf{f}](x) = x \qquad [\mathbf{g}](x) = x + 1$$

is a model of  $\mathcal{R}_{fg}$ , since for arbitrary  $x, y \in \mathbb{N}$ , we have

$$[\mathbf{f}](x) \geq x \qquad [\mathbf{g}](x) = x + 1 \geq y \vee [\mathbf{f}](x) = x \not\geq y$$

Then, with Theorem 3 we can show that  $\mathbf{f}(x) \rightarrow \mathbf{g}(x)$  is  $\mathcal{R}_{fg}$ -unsatisfiable, as  $[\mathbf{f}](x) = x \not\geq x + 1 = [\mathbf{g}](x)$  for every  $x \in \mathbb{N}$ .

To use  $\text{WPO}(\mathcal{A})$  in combination with Theorem 3, we need to validate formulas with predicate  $\sqsupseteq_{\text{WPO}(\mathcal{A})}$  in the term algebra  $\mathcal{T}(\mathcal{F}, \mathcal{V})$ . We encode these formulas into formulas with predicates  $\geq$  and  $>$ , which are then interpreted in  $\mathcal{A}$ .

**Definition 9 (formal WPO).** Let  $\langle \geq, > \rangle$  and  $\langle \lesssim, \succ \rangle$  be pairs of relations over some set and over  $\mathcal{F}$ , respectively, and let  $\pi$  be a partial status. We define  $\text{wpo}(\pi, \geq, >, \lesssim, \succ)$  or  $\text{wpo}$  for short to be the pair  $\langle \sqsupseteq_{\text{wpo}}, \sqsubset_{\text{wpo}} \rangle$ , where for terms  $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ ,  $s \sqsupseteq_{\text{wpo}} t$  and  $s \sqsubset_{\text{wpo}} t$  are Boolean formulas defined as follows:

$$s \sqsubset_{\text{wpo}} t := s > t \vee (s \geq t \wedge \phi)$$

where  $\phi$  is FALSE if  $s \in \mathcal{V}$  and is  $\bigvee_{i \in \pi(f)} s_i \sqsubseteq_{\mathbf{wpo}} t \vee \psi$  if  $s = f(s_1, \dots, s_n)$ , and  $\psi$  is FALSE if  $t \in \mathcal{V}$  and is

$$\bigwedge_{j \in \pi(g)} s \sqsubseteq_{\mathbf{wpo}} t_j \wedge (f \succ g \vee (f \lesssim g \wedge \pi_f(s_1, \dots, s_n) \sqsubseteq_{\mathbf{wpo}}^{\text{lex}} \pi_g(t_1, \dots, t_m)))$$

if  $t = g(t_1, \dots, t_m)$ . Formula  $s \sqsubseteq_{\mathbf{wpo}} t$  is defined analogously, except that  $\phi$  is TRUE if  $s = t \in \mathcal{V}$ , and  $\sqsubseteq_{\mathbf{wpo}}^{\text{lex}}$  in formula  $\psi$  is replaced by  $\sqsubseteq_{\mathbf{wpo}}$ .

We omit an easy proof that verifies that  $\mathbf{wpo}$  encodes WPO:

**Lemma 7.**  $s \sqsubseteq_{(\sqsubseteq)_{\mathbf{WPO}(\mathcal{A})}} t$  iff  $\mathcal{A} \models s \sqsubseteq_{\mathbf{wpo}} t$ .

Note carefully that  $s \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} t$  is  $\mathcal{A} \not\models s \sqsubseteq_{\mathbf{wpo}} t$  but not  $\mathcal{A} \models s \not\sqsubseteq_{\mathbf{wpo}} t$ . Hence we ensure  $s \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} t$  by  $\mathcal{A} \models s \sqsubseteq_{\overline{\mathbf{wpo}}} t$ , where  $\overline{\mathbf{wpo}}$  denotes  $\mathbf{wpo}(\pi, \not\prec, \not\leq, \not\prec, \not\leq)$ .

**Theorem 4.** If  $\mathcal{R}$  is a CTRS,  $\langle \geq, > \rangle$  and  $\langle \lesssim, \succ \rangle$  are order pairs on  $\mathcal{A}$  and  $\mathcal{F}$ ,  $\langle \mathcal{A}, \geq \rangle$  is  $\pi$ -simple and monotone,  $\mathcal{A} \models l \sqsubseteq_{\mathbf{wpo}} r \vee u_1 \sqsubseteq_{\overline{\mathbf{wpo}}} v_1 \vee \dots \vee u_n \sqsubseteq_{\overline{\mathbf{wpo}}} v_n$  for every  $(l \rightarrow r \leftarrow u_1 \rightarrow v_1, \dots, u_n \rightarrow v_n) \in \mathcal{R}$ , and  $\mathcal{A} \models s \sqsubseteq_{\overline{\mathbf{wpo}}} t$ , then  $s \rightarrow t$  is  $\mathcal{R}$ -unsatisfiable.

*Proof.* We apply Theorem 3. To this end, we first show that  $\langle \mathcal{T}(\mathcal{F}, \mathcal{V}), \sqsubseteq_{\mathbf{WPO}(\mathcal{A})} \rangle$  is a monotone preordered model of  $\mathcal{R}$ . Monotonicity and preorderedness are due to Proposition 1. For being a model, let  $(l \rightarrow r \leftarrow u_1 \rightarrow v_1, \dots, u_n \rightarrow v_n) \in \mathcal{R}$ . Due to assumption and Lemma 7, we have  $l \sqsubseteq_{\mathbf{WPO}(\mathcal{A})} r \vee u_1 \sqsubseteq_{\overline{\mathbf{WPO}(\mathcal{A})}} v_1 \vee \dots \vee u_n \sqsubseteq_{\overline{\mathbf{WPO}(\mathcal{A})}} v_n$ . Due to Lemmas 2 and 4, we get  $l\theta \sqsubseteq_{\mathbf{WPO}(\mathcal{A})} r\theta \vee u_1\theta \sqsubseteq_{\overline{\mathbf{WPO}(\mathcal{A})}} v_1\theta \vee \dots \vee u_n\theta \sqsubseteq_{\overline{\mathbf{WPO}(\mathcal{A})}} v_n\theta$  for every  $\theta : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ . With Proposition 2 we conclude  $\mathcal{T}(\mathcal{F}, \mathcal{V}) \models l \sqsubseteq_{\mathbf{WPO}(\mathcal{A})} r \vee u_1 \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} v_1 \vee \dots \vee u_n \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} v_n$ . Finally, we need  $\mathcal{T}(\mathcal{F}, \mathcal{V}) \models s \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} t$ , i.e.,  $s\theta \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} t\theta$  for any  $\theta : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ . As we assume  $s \sqsubseteq_{\overline{\mathbf{WPO}(\mathcal{A})}} t$ , by Lemma 4 we have  $s\theta \sqsubseteq_{\overline{\mathbf{WPO}(\mathcal{A})}} t\theta$ . By Proposition 2 we conclude  $s\theta \not\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} t\theta$ .  $\square$

## 7 Experiments

The proposed methods are implemented in the termination prover NaTT [35], available at <https://www.trs.cm.is.nagoya-u.ac.jp/NaTT/>.

Internally, NaTT reduces the problem of finding an algebra  $\mathcal{A}$  that make  $\langle \mathcal{A}, \geq \rangle$  a model of a TRS  $\mathcal{R}$  (or  $\sqsubseteq_{\mathbf{WPO}(\mathcal{A})} \subseteq \mathcal{R}$ ) into a satisfiability modulo theory (SMT) problem, which is then solved by the backend SMT solver z3 [26]. The implementation of Theorem 1 and Corollary 1 is a trivial adaptation from the termination methods. Corollary 2 is also trivial for totally ordered carriers, since  $\mathcal{A} \models s \not\leq t$  is equivalent to  $\mathcal{A} \models s < t$ . Matrix/tuple interpretations are also easy, since  $\mathcal{A} \models (a_1, \dots, a_n) \not\leq (b_1, \dots, b_n)$  is equivalent to  $\mathcal{A} \models a_1 < b_1 \vee \dots \vee a_n < b_n$ . Theorem 2 with WPO is obtained by parametrizing WPO.

**Table 1.** Experimental results.

Method	TRS						CTRS		
	$\mathcal{R}_{add}$	$\mathcal{R}_{mat}$	$\mathcal{R}_A$	$\mathcal{R}_{kbo}$	$\mathcal{R}_{wpo}$	COPS(15)	$\mathcal{R}_{ab}$	$\mathcal{R}_{fg}$	COPS(126)
$Sum$						6		✓	15
$Sum^+$						6		✓	24
$Sum^-$	✓					5			28
$Mat$		✓				6	✓	✓	25 (TO:88)
LPO			✓			5	✓		19
$WPO(Sum)$				✓	✓	6	✓	✓	15
$WPO(Sum^+)$				✓	✓	6	✓	✓	25 (TO:29)
$WPO(Sum^-)$						6	✓		15
infChecker	✓		✓			13	✓	✓	51+42 (TO:25)
CO3		✓				5	✓		20
NaTT 2.1						3			19
NaTT 2.2	✓	✓	✓	✓	✓	6 (TO:4)	✓	✓	31 (TO:79)

Theorem 3 needs some tricks. In the unconditional case, finding a desired algebra  $\mathcal{A}$  can be encoded into SMT over quantifier-free linear arithmetic for a large class of  $\mathcal{A}$  [36]. For the conditional case, we need to find  $(\exists)$  parameters that validates  $(\forall)$  a disjunctive clause. Farkas' lemma would reduce such a problem into quantifier-free SMT, but then the resulting problem is nonlinear. Experimentally, we observe that our backend **z3** performs better on quantified linear arithmetic than quantifier-free nonlinear arithmetic, and hence we choose to leave the  $\forall$  quantifiers.

We conducted experiments using the examples presented in the paper and the examples in the INF category of the standard benchmark set COPS. The execution environment is StarExec [31] with the same settings as CoCo 2019.

Many COPS examples contain conjunctive reachability constraints of form  $s_1 \rightarrow t_1 \wedge \dots \wedge s_n \rightarrow t_n$ . In this experiment we naively collapsed such a constraint into  $\mathbf{tp}(s_1, \dots, s_n) \rightarrow \mathbf{tp}(t_1, \dots, t_n)$  by introducing a fresh function symbol **tp**. Two benchmarks exceed the scope of oriented CTRSs, on which NaTT immediately gives up.

As co-rewrite pairs we tested algebras  $Sum$ ,  $Sum^+$ ,  $Sum^-$ ,  $Mat$ , LPO, and WPO. The basic algebra  $Sum = \langle \mathbb{Z}, [\cdot] \rangle$  is given by  $[f](x_1, \dots, x_n) = c_0 + \sum_{i=1}^n c_i \cdot x_i$ , where  $c_0 \in \mathbb{Z}$ ,  $c_1, \dots, c_n \in \{0, 1\}$ . Similarly  $Sum^+$  and  $Sum^-$  are defined, where the ranges of  $c_0$ , which also determine the carrier, are  $\mathbb{N}$  and  $\mathbb{Z}_{\leq 0}$ , respectively. The algebra  $Mat$  represents the 2D matrix interpretations.

Table 1 presents the results. For TRSs, we can observe that our proposed methods advance the state of the art, in the sense that they prove new examples that no tool previously participated in CoCo could handle. As there are only 15 TRS examples in the INF category of COPS 2021, we could not derive interesting observations there. Taking CTRS examples into account, we see  $Sum$  is not as

good as  $\text{Sum}^+$  or  $\text{Sum}^-$ , while the carrier is bigger ( $\mathbb{Z}$  versus  $\mathbb{N}$  or  $\mathbb{Z}_{\leq 0}$ ). This phenomenon is explained as follows: For the latter two one knows variables are bounded by 0 (from below or above), and hence one can have  $\text{Sum}^+ \models x \geq a$  or  $\text{Sum}^- \models a \geq x$  by  $[a] = 0$ . Neither is possible when the carrier is unbounded. This observation also suggests another choice of carriers that are bounded from below and above, which is left for future work.

From the figures in CTRS examples,  $\text{Sum}^-$  performs the best among our methods. However,  $\text{Mat}$  and  $\text{WPO}(\text{Sum}^+)$  solve more examples if TRS examples are counted. It does not seem appropriate yet to judge practical significance from these experiments.

Finally, we implemented as the default strategy of NaTT 2.2 the sequential application of  $\text{Sum}^-$ , LPO,  $\text{WPO}(\text{Sum}^+)$ , and  $\text{Mat}$  after the test NaTT already have implemented. There improvement over previous NaTT 2.1 should be clear, although the number of timeouts (indicated by “TO:”) is significant.

## 8 Conclusion

We proposed generalizations of termination techniques that can prove unsatisfiability of reachability, both for term rewriting and for conditional term rewriting. We implemented the approach in the termination prover NaTT, and experimentally evaluated the significance of the proposed approach.

The implementation focused on evaluating the proposed methods separately. The only implemented way of combining their power is a naive one: apply the tests one by one while they fail. For future work, it will be interesting to incorporate the proposed method into the existing frameworks [10, 30].

**Acknowledgments.** The author would like to thank Kiraku Shintani for the technical help with the COPS database system. I would also like to thank Nao Hirokawa, Salvador Lucas, Naoki Nishida, and Sarah Winkler for discussions, and the anonymous reviewers for their detailed comments that improved the presentation of the paper.

## References

1. Aoto, T.: Disproving confluence of term rewriting systems by interpretation and ordering. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) FroCoS 2013. LNCS (LNAI), vol. 8152, pp. 311–326. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40885-4\\_22](https://doi.org/10.1007/978-3-642-40885-4_22)
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theor. Comput. Sci. **236**(1–2), 133–178 (2000). [https://doi.org/10.1016/S0304-3975\(99\)00207-8](https://doi.org/10.1016/S0304-3975(99)00207-8)
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
4. Dershowitz, N.: Orderings for term-rewriting systems. Theor. Comput. Sci. **17**(3), 279–301 (1982). [https://doi.org/10.1016/0304-3975\(82\)90026-3](https://doi.org/10.1016/0304-3975(82)90026-3)
5. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. J. Autom. Reason. **40**(2–3), 195–220 (2008). <https://doi.org/10.1007/s10817-007-9087-9>

6. Feuillade, G., Genet, T., Tong, V.V.T.: Reachability analysis over term rewriting systems. *J. Autom. Reasoning* **33**, 341–383 (2004). <https://doi.org/10.1007/s10817-004-6246-0>
7. Giesl, J., et al.: Proving termination of programs automatically with AProVE. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) *IJCAR 2014*. LNCS (LNAI), vol. 8562, pp. 184–191. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08587-6\\_13](https://doi.org/10.1007/978-3-319-08587-6_13)
8. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) *TACAS 2019*. LNCS, vol. 11429, pp. 156–166. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17502-3\\_10](https://doi.org/10.1007/978-3-030-17502-3_10)
9. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. *J. Autom. Reason.* **37**(3), 155–203 (2006). <https://doi.org/10.1007/s10817-006-9057-7>
10. Gutiérrez, R., Lucas, S.: Automatically proving and disproving feasibility conditions. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) *IJCAR 2020*. LNCS (LNAI), vol. 12167, pp. 416–435. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51054-1\\_27](https://doi.org/10.1007/978-3-030-51054-1_27)
11. Gutiérrez, R., Lucas, S.: MU-TERM: verify termination properties automatically (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) *IJCAR 2020*. LNCS (LNAI), vol. 12167, pp. 436–447. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-51054-1\\_28](https://doi.org/10.1007/978-3-030-51054-1_28)
12. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. *Inf. Comput.* **199**(1–2), 172–199 (2005). <https://doi.org/10.1016/j.ic.2004.10.004>
13. Kamin, S., Lévy, J.J.: Two generalizations of the recursive path ordering (1980). unpublished note
14. Knuth, D.E., Bendix, P.: Simple word problems in universal algebras. In: *Computational Problems in Abstract Algebra*, pp. 263–297. Pergamon Press, New York (1970). <https://doi.org/10.1016/B978-0-08-012975-4.50028-X>
15. Kop, C., Vale, D.: Tuple interpretations for higher-order complexity. In: Kobayashi, N. (ed.) *FSCD 2021*. LIPIcs, vol. 195, pp. 31:1–31:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). <https://doi.org/10.4230/LIPIcs.FSCD.2021.31>
16. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean termination tool 2. In: Treinen, R. (ed.) *RTA 2009*. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02348-4\\_21](https://doi.org/10.1007/978-3-642-02348-4_21)
17. Kusakari, K., Nakamura, M., Toyama, Y.: Argument filtering transformation. In: Nadathur, G. (ed.) *PPDP 1999*. LNCS, vol. 1702, pp. 47–61. Springer, Heidelberg (1999). [https://doi.org/10.1007/10704567\\_3](https://doi.org/10.1007/10704567_3)
18. Lankford, D.: Canonical algebraic simplification in computational logic. Technical report ATP-25, University of Texas (1975)
19. Lucas, S., Gutiérrez, R.: Use of logical models for proving infeasibility in term rewriting. *Inf. Process. Lett.* **136**, 90–95 (2018). <https://doi.org/10.1016/j.ipl.2018.04.002>
20. Lucas, S., Meseguer, J.: 2D dependency pairs for proving operational termination of CTRSs. In: Escobar, S. (ed.) *WRLA 2014*. LNCS, vol. 8663, pp. 195–212. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-12904-4\\_11](https://doi.org/10.1007/978-3-319-12904-4_11)
21. Lucas, S., Meseguer, J.: Dependency pairs for proving termination properties of conditional term rewriting systems. *J. Log. Algebraic Methods Program.* **86**(1), 236–268 (2017). <https://doi.org/10.1016/j.jlamp.2016.03.003>

22. Lucas, S., Meseguer, J., Gutiérrez, R.: The 2D dependency pair framework for conditional rewrite systems. part I: definition and basic processors. *J. Comput. Syst. Sci.* **96**, 74–106 (2018). <https://doi.org/10.1016/j.jcss.2018.04.002>
23. Manna, Z., Ness, S.: Termination of Markov algorithms (1969). unpublished manuscript
24. Middeldorp, A.: Approximating dependency graphs using tree automata techniques. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) *IJCAR 2001*. LNCS, vol. 2083, pp. 593–610. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-45744-5\\_49](https://doi.org/10.1007/3-540-45744-5_49)
25. Middeldorp, A., Nagele, J., Shintani, K.: CoCo 2019: report on the eighth confluence competition. *Int. J. Softw. Tools Technol. Transfer* **23**(6), 905–916 (2021). <https://doi.org/10.1007/s10009-021-00620-4>
26. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) *TACAS 2008*. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_24](https://doi.org/10.1007/978-3-540-78800-3_24)
27. Ohlebusch, E.: *Advanced Topics in Term Rewriting*. Springer, New York (2002). <https://doi.org/10.1007/978-1-4757-3661-8>
28. Oostrom, V.: Sub-Birkhoff. In: Kameyama, Y., Stuckey, P.J. (eds.) *FLOPS 2004*. LNCS, vol. 2998, pp. 180–195. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-24754-8\\_14](https://doi.org/10.1007/978-3-540-24754-8_14)
29. Sternagel, C., Thiemann, R., Yamada, A.: A formalization of weighted path orders and recursive path orders. *Arch. Formal Proofs* (2021). [https://isa-afp.org/entries/Weighted\\_Path\\_Order.html](https://isa-afp.org/entries/Weighted_Path_Order.html), Formal proof development
30. Sternagel, C., Yamada, A.: Reachability analysis for termination and confluence of rewriting. In: Vojnar, T., Zhang, L. (eds.) *TACAS 2019*. LNCS, vol. 11427, pp. 262–278. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17462-0\\_15](https://doi.org/10.1007/978-3-030-17462-0_15)
31. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: a cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) *IJCAR 2014*. LNCS (LNAI), vol. 8562, pp. 367–373. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08587-6\\_28](https://doi.org/10.1007/978-3-319-08587-6_28)
32. TeReSe: Term Rewriting Systems, Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
33. Thiemann, R., Schöpf, J., Sternagel, C., Yamada, A.: Certifying the weighted path order (invited talk). In: Ariola, Z.M. (ed.) *FSCD 2020*. LIPIcs, vol. 167, pp. 4:1–4:20. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). <https://doi.org/10.4230/LIPIcs.FSCD.2020.4>
34. Yamada, A.: Multi-dimensional interpretations for termination of term rewriting. In: Platzer, A., Sutcliffe, G. (eds.) *CADE 2021*. LNCS (LNAI), vol. 12699, pp. 273–290. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_16](https://doi.org/10.1007/978-3-030-79876-5_16)
35. Yamada, A., Kusakari, K., Sakabe, T.: Nagoya termination tool. In: Dowek, G. (ed.) *RTA 2014*. LNCS, vol. 8560, pp. 466–475. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08918-8\\_32](https://doi.org/10.1007/978-3-319-08918-8_32)
36. Yamada, A., Kusakari, K., Sakabe, T.: A unified ordering for termination proving. *Sci. Comput. Program.* **111**, 110–134 (2015). <https://doi.org/10.1016/j.scico.2014.07.009>
37. Zantema, H.: Termination of term rewriting by semantic labelling. *Fundam. Informaticae* **24**(1/2), 89–105 (1995). <https://doi.org/10.3233/FI-1995-24124>
38. Zantema, H.: The termination hierarchy for term rewriting. *Appl. Algebr. Eng. Comm. Comput.* **12**(1/2), 3–19 (2001). <https://doi.org/10.1007/s002000100061>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# **Knowledge Representation and Justification**



# Evonne: Interactive Proof Visualization for Description Logics (System Description)

Christian Alrabbaa<sup>1</sup>(✉)(iD), Franz Baader<sup>1</sup>(✉)(iD), Stefan Borgwardt<sup>1</sup>(✉)(iD),  
Raimund Dachzelt<sup>2</sup>(✉)(iD), Patrick Koopmann<sup>1</sup>(✉)(iD), and Julián Méndez<sup>2</sup>(✉)(iD)

<sup>1</sup> Institute of Theoretical Computer Science, TU Dresden, Dresden, Germany  
{christian.alrabbaa, franz.baader, stefan.borgwardt,  
patrick.koopmann}@tu-dresden.de

<sup>2</sup> Interactive Media Lab Dresden, TU Dresden, Dresden, Germany  
{raimund.dachzelt, julian.mendez2}@tu-dresden.de

**Abstract.** Explanations for description logic (DL) entailments provide important support for the maintenance of large ontologies. The “justifications” usually employed for this purpose in ontology editors pinpoint the parts of the ontology responsible for a given entailment. Proofs for entailments make the intermediate reasoning steps explicit, and thus explain how a consequence can actually be derived. We present an interactive system for exploring description logic proofs, called EVONNE, which visualizes proofs of consequences for ontologies written in expressive DLs. We describe the methods used for computing those proofs, together with a feature called *signature-based proof condensation*. Moreover, we evaluate the quality of generated proofs using real ontologies.

## 1 Introduction

Proofs generated by Automated Reasoning (AR) systems are sometimes presented to humans in textual form to convince them of the correctness of a theorem [9, 11], but more often employed as certificates that can automatically be checked [20]. In contrast to the AR setting, where very long proofs may be needed to derive a deep mathematical theorem from very few axioms, DL-based ontologies are often very large, but proofs of a single consequence are usually of a more manageable size. For this reason, the standard method of explanation in description logic [8] has long been to compute so-called *justifications*, which point out a minimal set of source statements responsible for an entailment of interest. For example, the ontology editor Protégé<sup>1</sup> supports the computation of justifications since 2008 [12], which is very useful when working with large DL ontologies. Nevertheless, it is often not obvious why a given consequence actually follows from such a justification [13]. Recently, this explanation capability has been extended towards showing full *proofs* with intermediate reasoning steps, but this is restricted to ontologies written in the lightweight DLs supported by the ELK reasoner [15, 16], and the graphical presentation of proofs is very basic.

<sup>1</sup> <https://protege.stanford.edu/>.

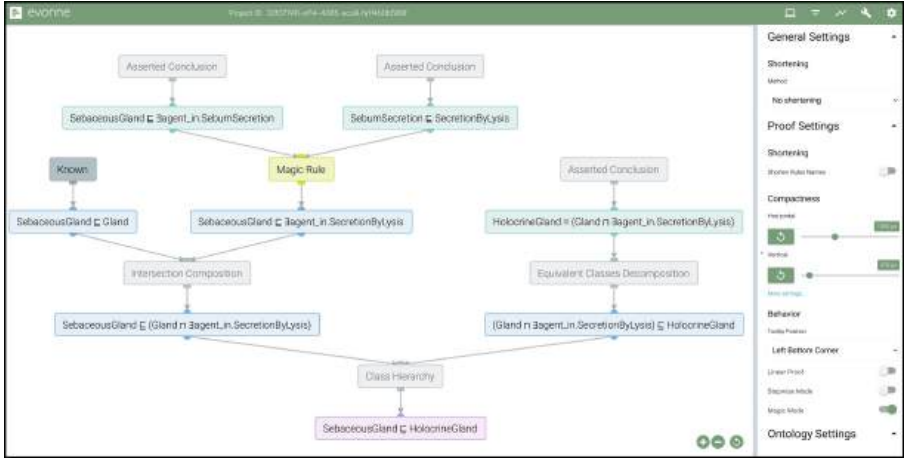
In this paper, we present EVONNE as an interactive system, for exploring DL proofs for description logic entailments, using the methods for computing small proofs presented in [3, 5]. Initial prototypes of EVONNE were presented in [6, 10], but since then, many improvements were implemented. While EVONNE does more than just visualizing proofs, this paper focuses on the proof component of EVONNE: specifically, we give a brief overview of the interface for exploring proofs, describe the proof generation methods implemented in the back-end, and present an experimental evaluation of these proofs generation methods in terms of proof size and run time. The improved back-end uses Java libraries that extract proofs using various methods, such as from the ELK calculus, or *forgetting-based proofs* [3] using the forgetting tools LETHE [17] and FAME [21] in a black-box fashion. The new front-end is visually more appealing than the prototypes presented in [6, 10], and allows to inspect and explore proofs using various interaction techniques, such as zooming and panning, collapsing and expanding, text manipulation, and compactness adjustments. Additional features include the minimization of the generated proofs according to various measures and the possibility to select a *known signature* that is used to automatically hide parts of the proofs that are assumed to be obvious for users with certain previous knowledge. Our evaluation shows that proof sizes can be significantly reduced in this way, making the proofs more user-friendly. EVONNE can be tried and downloaded at <https://imld.de/evonne>. The version of EVONNE described here, as well as the data and scripts used in our experiments, can be found at [2].

## 2 Preliminaries

We recall some relevant notions for DLs; for a detailed introduction, see [8]. DLs are decidable fragments of first-order logic (FOL) with a special, variable-free syntax, and that use only unary and binary predicates, called *concept names* and *role names*, respectively. These can be used to build complex *concepts*, which correspond to first-order formulas with one free variable, and *axioms* corresponding to first-order sentences. Which kinds of concepts and axioms can be built depends on the expressivity of the used DL. Here we mainly consider the light-weight DL  $\mathcal{ELH}$  and the more expressive  $\mathcal{ALCH}$ . We have the usual notion of FOL *entailment*  $\mathcal{O} \models \alpha$  of an axiom  $\alpha$  from a finite set of axioms  $\mathcal{O}$ , called an ontology. of special interest are entailments of *atomic CIs* (concept inclusions) of the form  $A \sqsubseteq B$ , where  $A$  and  $B$  are concept names. Following [3], we define *proofs* of  $\mathcal{O} \models \alpha$  as finite, acyclic, directed hypergraphs, where vertices  $v$  are labeled with axioms  $\ell(v)$  and hyperedges are of the form  $(S, d)$ , with  $S$  a set of vertices and  $d$  a vertex such that  $\{\ell(v) \mid v \in S\} \models \ell(d)$ ; the leaves of a proof must be labeled by elements of  $\mathcal{O}$  and the root by  $\alpha$ . In this paper, all proofs are *trees*, i.e. no vertex can appear in the first component of multiple hyperedges (see Fig. 1).

## 3 The Graphical User Interface

The user interface of EVONNE is implemented as a web application. To support users in understanding large proofs, they are offered various layout options and











**Fig. 1.** Overview of EVONNE - a condensed proof in the bidirectional layout

interaction components. The proof visualization is linked to a second view showing the context of the proof in a relevant subset of the ontology. In this ontology view, interactions between axioms are visualized, so that users can understand the context of axioms occurring in the proof. The user can also examine possible ways to eliminate unwanted entailments in the ontology view. The focus of this system description, however, is on the proof component: we describe how the proofs are generated and how users can interact with the proof visualization. For details on the ontology view, we refer the reader to the workshop paper [6], where we also describe how EVONNE supports ontology repair.

*Initialization.* After starting EVONNE for the first time, users create a new project, for which they specify an ontology file. They can then select an entailed atomic CI to be explained. The user can choose between different proof methods, and optionally select a signature of *known terms* (cf. Sect. 4), which can be generated using the term selection tool Protégé-TS [14].

*Layout.* Proofs are shown as graphs with two kinds of vertices: colored vertices for axioms, gray ones for inference steps. By default, proofs are shown using a *tree layout*. To take advantage of the width of the display when dealing with long axioms, it is possible to show proofs in a *vertical layout*, placing axioms linearly below each other, with inferences represented through edges on the side (without the inference vertices). It is possible to automatically re-order vertices to minimize the distance between conclusion and premises in each step. The third layout option is the *bidirectional layout* (see Fig. 1), a tree layout where, initially, the entire proof is collapsed into a *magic vertex* that links the conclusion directly to its justification, and from which individual inference steps can be pulled out and pushed back from both directions.

*Exploration.* In all views, each vertex is equipped with multiple functionalities for exploring a proof. For proofs generated with ELK, clicking on an inference vertex shows the inference rule used, and the particular inference with relevant sub-elements highlighted in different colors. Axiom vertices show different button (, , , ) when hovered over. In the standard tree layout, users can hide sub-proofs under an axiom () or reveal the previous inference step () or the entire-sub-proof (). In the vertical layout, the button () highlights and explains the inference of the current axiom. In the bidirectional layout, the arrow buttons are used for pulling inference steps out of the magic vertex, as well as pushing them back in.

*Presentation.* A *minimap* allows users to keep track of the overall structure of the proof, thus enriching the zooming and panning functionality. Users can adjust width and height of proofs through the options side-bar. Long axiom labels can be *shortened* in two ways: either by setting a fixed size to all vertices, or by abbreviating names based on capital letters. Afterwards, it is possible to restore the original labels individually.

## 4 Proof Generation

To obtain the proofs that are shown to the user, we implemented different proof generation techniques, some of which were initially described in [3]. For  $\mathcal{ELH}$  ontologies, proofs can be generated natively by the DL reasoner ELK [16]. These proofs use rules from the calculus described in [16]. We apply the Dijkstra-like algorithm introduced in [4, 5] to compute a *minimized proof* from the ELK output. This minimization can be done w.r.t. different measures, such as the size, depth, or weighted sum (where each axiom is weighted by its size), as long as they are *monotone* and *recursive* [5]. For ontologies outside of the  $\mathcal{ELH}$  fragment, we use the forgetting-based approach originally described in [3], for which we now implemented two alternative algorithms for computing more compact proofs (Sect. 4.1). Finally, independently of the proof generation method, one can specify a signature of known terms. This signature contains terminology that the user is familiar with, so that entailments using only those terms do not need to be explained. The condensation of proofs w.r.t. signatures is described in Sect. 4.2.

### 4.1 Forgetting-Based Proofs

In a forgetting-based proof, proof steps represent inferences on concept or role names using a *forgetting* operation. Given an ontology  $\mathcal{O}$  and a predicate name  $x$ , the result  $\mathcal{O}^{-x}$  of forgetting  $x$  in  $\mathcal{O}$  does not contain any occurrences of  $x$ , while still capturing all entailments of  $\mathcal{O}$  that do not use  $x$  [18]. In a forgetting-based proof, an inference takes as premises a set  $\mathcal{P}$  of axioms and has as conclusion some axiom  $\alpha \in \mathcal{P}^{-x}$  (where a particular forgetting operation is used to compute  $\mathcal{P}^{-x}$ ). Intuitively,  $\alpha$  is obtained from  $\mathcal{P}$  by performing inferences on  $x$ . To

compute a forgetting-based proof, we have to forget the names occurring in the ontology one after the other, until only the names occurring in the statement to be proved are left. For the forgetting operation, the user can select between two implementations: LETHE [17] (using the method supporting  $\mathcal{ALCH}$ ) and FAME [21] (using the method supporting  $\mathcal{ALCOI}$ ). Since the space of possible inference steps is exponentially large, it is not feasible to minimize proofs after their computation, as we do for  $\mathcal{EL}$  entailments, which is why we rely on heuristics and search algorithms to generate small proofs. Specifically, we implemented three methods for computing forgetting-based proofs: **HEUR** tries to find proofs fast, **SYMB** tries to minimize the number of predicates forgotten in a proof, with the aim of obtaining proofs of small depth, and **SIZE** tries to optimize the size of the proof. The heuristic method **HEUR** is described in [3], and its implementation has not been changed since then. The search methods **SYMB** and **SIZE** are new (details can be found in the extended version [1]).

## 4.2 Signature-Based Proof Condensation

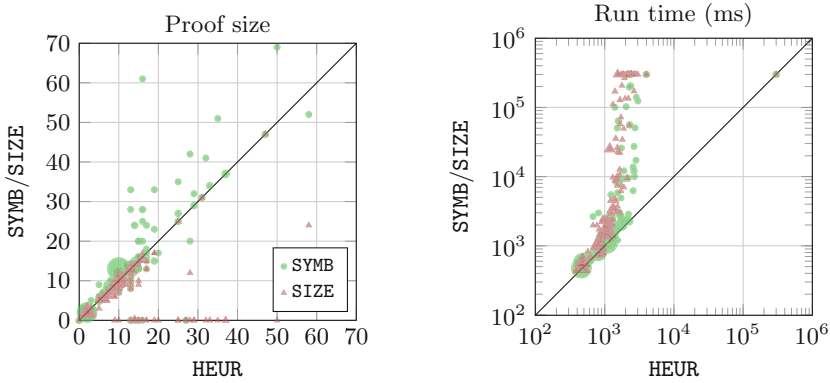
When inspecting a proof over a real-world ontology, different parts of the proof will be more or less familiar to the user, depending on their knowledge about the involved concepts or their experience with similar inference steps in the past. For CIs between concepts for which a user has application knowledge, they may not need to see a proof, and consequently, sub-proofs for such axioms can be automatically hidden. We assume that the user’s knowledge is given in the form of a *known signature*  $\Sigma$  and that axioms that contain only symbols from  $\Sigma$  do not need to be explained. The effect can be seen in Fig. 1 through the “known”-inference on the left, where  $\Sigma$  contains **SebaceousGland** and **Gland**. The known signature is taken into consideration when minimizing the proofs, so that proofs are selected for which more of the known information can be used if convenient. This can be easily integrated into the Dijkstra approach described in [3], by initially assigning to each axiom covered by  $\Sigma$  a proof with a single vertex.

## 5 Evaluation

For EVONNE to be usable in practice, it is vital that proofs are computed efficiently and that they are not too large. An experimental evaluation of minimized proofs for  $\mathcal{EL}$  and forgetting-based proofs obtained with FAME and LETHE is provided in [3]. We here present an evaluation of additional aspects: 1) a comparison of the three methods for computing forgetting-based proofs, and 2) an evaluation on the impact of signature-based proof condensation. All experiments were performed on Debian Linux (Intel Core i5-4590, 3.30 GHz, 23 GB Java heap size).

### 5.1 Minimal Forgetting-Based Proofs

To evaluate forgetting-based proofs, we extracted  $\mathcal{ALCH}$  “proof tasks” from the ontologies in the 2017 snapshot of BioPortal [19]. We restricted all ontologies



**Fig. 2.** Run times and proof sizes for different forgetting-based proof methods. Marker size indicates how often each pattern occurred in the BioPortal snapshot. Instances that timed out were assigned size 0.

to  $\mathcal{ALCH}$  and collected all entailed atomic CIs  $\alpha$ , for each of which we computed the union  $\mathcal{U}$  of all their justifications. We identified pairs  $(\alpha, \mathcal{U})$  that were isomorphic modulo renaming of predicates, and kept only those patterns  $(\alpha, \mathcal{U})$  that contained at least one axiom not expressible in  $\mathcal{ELH}$ . This was successful in 373 of the ontologies<sup>2</sup> and resulted in 138 distinct *justification patterns*  $(\alpha, \mathcal{U})$ , representing 327 different entailments in the BioPortal snapshot. We then computed forgetting-based proofs for  $\mathcal{U} \models \alpha$  with our three methods using LETHE, with a 5-minute timeout. This was successful for 325/327 entailments for the heuristic method (HEUR), 317 for the symbol-minimizing method (SYMB), and 279 for the size-minimizing method (SIZE). In Fig. 2 we compare the resulting *proof sizes* (left) and the *run times* (right), using HEUR as baseline (x-axis). HEUR is indeed faster in most cases, but SIZE reduces proof size by 5% on average compared to HEUR, which is not the case for SYMB. Regarding *proof depth* (not shown in the figure), SYMB did not outperform HEUR on average, while SIZE surprisingly yielded an average reduction of 4% compared to HEUR. Despite this good performance of SIZE for proof size and depth, for entailments that depend on many or complex axioms, computation times for both SYMB and SIZE become unacceptable, while proof generation with HEUR mostly stays in the area of seconds.

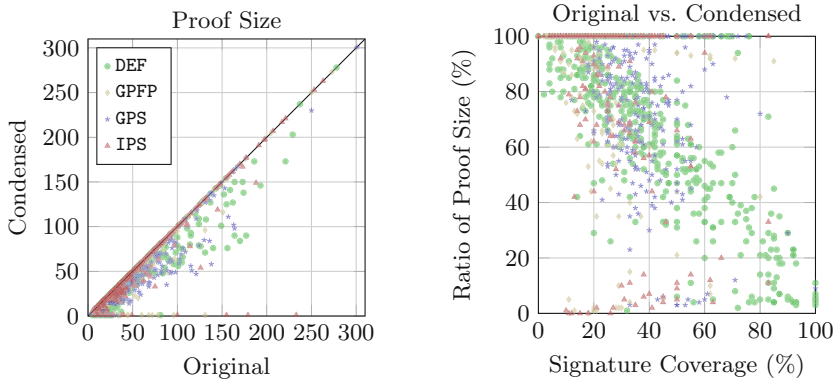
## 5.2 Signature-Based Proof Condensation

To evaluate how much hiding proof steps in a known signature decreases proof size in practice, we ran experiments on the large medical ontology SNOMED CT (International Edition, July 2020) that is mostly formulated in  $\mathcal{ELH}$ .<sup>3</sup> As signatures we used SNOMED CT *Reference Sets*,<sup>4</sup> which are restricted vocabularies

<sup>2</sup> The other ontologies could not be processed in this way within the memory limit.

<sup>3</sup> <https://www.snomed.org/>.

<sup>4</sup> <https://confluence.ihtsdotools.org/display/DOCRFSPG/2.3.+Reference+Set>.



**Fig. 3.** Size of original and condensed proofs (left). Ratio of proof size depending on the signature coverage (right).

for specific use cases. We extracted justifications similarly to the previous experiment, but did not rename predicates and considered only proof tasks that use at least 5 symbols from the signature, since otherwise no improvement can be expected by using the signatures. For each signature, we randomly selected 500 out of 6.689.452 *proof tasks* (if at least 500 existed). This left the 4 reference sets *General Practitioner/Family Practitioner* (GPFP), *Global Patient Set* (GPS), *International Patient Summary* (IPS), and the one included in the SNOMED CT distribution (DEF). For each of the resulting 2.000 proof tasks, we used ELK [16] and our proof minimization approach to obtain (a) a proof of minimal size and (b) a proof of minimal size after hiding the selected signature. The distribution of proof sizes can be seen in Fig. 3. In 770/2.000 cases, a smaller proof was generated when using the signature. In 91 of these cases, the size was even be reduced to 1, i.e. the target axiom used only the given signature and therefore nothing else needed to be shown. In the other 679 cases with reduced size, the average *ratio* of reduced size to original size was 0.68–0.93 (depending on the signature). One can see that this ratio is correlated with the *signature coverage* of the original proof (i.e. the ratio of signature symbols to total symbols in the proof), with a weak or strong correlation depending on the signature ( $r$  between  $-0.26$  and  $-0.74$ ). However, a substantial number of proofs with relatively high signature coverage could still not be reduced in size at all (see the top right of the right diagram). In summary, we can see that signature-based condensation can be useful, but this depends on the proof task and the signature. We also conducted experiments on the Galen ontology,<sup>5</sup> with comparable results (see the extended version of this paper [1]).

<sup>5</sup> <https://bioportal.bioontology.org/ontologies/GALEN>.



## 6 Conclusion

We have presented and compared the proof generation and presentation methods used in EVONNE, a visual tool for explaining entailments of DL ontologies. While these methods produce smaller or less deep proofs, which are thus easier to present, there is still room for improvements. Specifically, as the forgetting-based proofs do not provide the same degree of detail as the ELK proofs, it would be desirable to also support methods for more expressive DLs that generate proofs with smaller inference steps. Moreover, our current evaluation focuses on proof size and depth—to understand how well EVONNE helps users to understand DL entailments, we would also need a qualitative evaluation of the tool with potential end-users. We are also working on explanations for non-entailments using countermodels [7] and a plugin for the ontology editor Protégé that is compatible with the PULi library and Proof Explanation plugin presented in [15], which will support all proof generation methods discussed here and more.<sup>6</sup>

**Acknowledgements.** This work was supported by the German Research Foundation (DFG) in Germany’s Excellence Strategy: EXC-2068, 390729961 - Cluster of Excellence “Physics of Life” and EXC 2050/1, 390696704 - Cluster of Excellence “Centre for Tactile Internet” (CeTI) of TU Dresden, by DFG grant 389792660 as part of TRR 248 - CPEC, by the AI competence center ScaDS.AI Dresden/Leipzig, and the DFG Research Training Group QuantLA, GRK 1763.

## References

1. Alrabbaa, C., Baader, F., Borgwardt, S., Dachselt, R., Koopmann, P., Méndez, J.: Evonne: interactive proof visualization for description logics (system description) - extended version (2022). <https://doi.org/10.48550/ARXIV.2205.09583>
2. Alrabbaa, C., Baader, F., Borgwardt, S., Dachselt, R., Koopmann, P., Méndez, J.: Evonne: interactive proof visualization for description logics (system description) - IJCAR22 - resources, May 2022. <https://doi.org/10.5281/zenodo.6560603>
3. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Finding small proofs for description logic entailments: theory and practice. In: Albert, E., Kovács, L. (eds.) Proceedings of the 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2020). EPiC Series in Computing, vol. 73, pp. 32–67. EasyChair (2020). <https://doi.org/10.29007/nhpp>
4. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: On the complexity of finding good proofs for description logic entailments. In: Borgwardt, S., Meyer, T. (eds.) Proceedings of the 33rd International Workshop on Description Logics (DL 2020). CEUR Workshop Proceedings, vol. 2663. CEUR-WS.org (2020). <http://ceur-ws.org/Vol-2663/paper-1.pdf>
5. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Finding good proofs for description logic entailments using recursive quality measures. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 291–308. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79876-5\\_17](https://doi.org/10.1007/978-3-030-79876-5_17)

<sup>6</sup> <https://github.com/de-tu-dresden-inf-lat/evee>.

6. Alrabbaa, C., Baader, F., Dachzelt, R., Flemisch, T., Koopmann, P.: Visualising proofs and the modular structure of ontologies to support ontology repair. In: Borgwardt, S., Meyer, T. (eds.) Proceedings of the 33rd International Workshop on Description Logics (DL 2020). CEUR Workshop Proceedings, vol. 2663. CEUR-WS.org (2020). <http://ceur-ws.org/Vol-2663/paper-2.pdf>
7. Alrabbaa, C., Hieke, W., Turhan, A.: Counter model transformation for explaining non-subsumption in  $\mathcal{EL}$ . In: Beierle, C., Ragni, M., Stolzenburg, F., Thimm, M. (eds.) Proceedings of the 7th Workshop on Formal and Cognitive Reasoning. CEUR Workshop Proceedings, vol. 2961, pp. 9–22. CEUR-WS.org (2021). [http://ceur-ws.org/Vol-2961/paper\\_2.pdf](http://ceur-ws.org/Vol-2961/paper_2.pdf)
8. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017). <https://doi.org/10.1017/9781139025355>
9. Fiedler, A.: Natural language proof explanation. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS (LNAI), vol. 2605, pp. 342–363. Springer, Heidelberg (2005). [https://doi.org/10.1007/978-3-540-32254-2\\_20](https://doi.org/10.1007/978-3-540-32254-2_20)
10. Flemisch, T., Langner, R., Alrabbaa, C., Dachzelt, R.: Towards designing a tool for understanding proofs in ontologies through combined node-link diagrams. In: Ivanova, V., Lambrix, P., Pesquita, C., Wiens, V. (eds.) Proceedings of the Fifth International Workshop on Visualization and Interaction for Ontologies and Linked Data (VOILA 2020). CEUR Workshop Proceedings, vol. 2778, pp. 28–40. CEUR-WS.org (2020). <http://ceur-ws.org/Vol-2778/paper3.pdf>
11. Horacek, H.: Presenting proofs in a human-oriented way. In: CADE 1999. LNCS (LNAI), vol. 1632, pp. 142–156. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48660-7\\_10](https://doi.org/10.1007/3-540-48660-7_10)
12. Horridge, M., Parsia, B., Sattler, U.: Explanation of OWL entailments in Protege 4. In: Bizer, C., Joshi, A. (eds.) Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC 2008). CEUR Workshop Proceedings, vol. 401. CEUR-WS.org (2008). <http://ceur-ws.org/Vol-401/iswc2008pd.submission.47.pdf>
13. Horridge, M., Parsia, B., Sattler, U.: Justification oriented proofs in OWL. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010. LNCS, vol. 6496, pp. 354–369. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-17746-0\\_23](https://doi.org/10.1007/978-3-642-17746-0_23)
14. Hyland, I., Schmidt, R.A.: Protégé-TS: An OWL ontology term selection tool. In: Borgwardt, S., Meyer, T. (eds.) Proceedings of the 33rd International Workshop on Description Logics (DL 2020). CEUR Workshop Proceedings, vol. 2663. CEUR-WS.org (2020). <http://ceur-ws.org/Vol-2663/paper-12.pdf>
15. Kazakov, Y., Klinov, P., Stupnikov, A.: Towards reusable explanation services in protege. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) Proceedings of the 30th International Workshop on Description Logics (DL 2017). CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017). <http://ceur-ws.org/Vol-1879/paper31.pdf>
16. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK - from polynomial procedures to efficient reasoning with  $\mathcal{EL}$  ontologies. J. Autom. Reason. **53**(1), 1–61 (2014). <https://doi.org/10.1007/s10817-013-9296-3>
17. Koopmann, P.: LETHE: forgetting and uniform interpolation for expressive description logics. Künstliche Intell. **34**(3), 381–387 (2020). <https://doi.org/10.1007/s13218-020-00655-w>

18. Koopmann, P., Schmidt, R.A.: Forgetting concept and role symbols in  $\mathcal{ALCH}$ -ontologies. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 552–567. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-45221-5\\_37](https://doi.org/10.1007/978-3-642-45221-5_37)
19. Matentzoglou, N., Parsia, B.: Biportal snapshot 30.03.2017, March 2017. <https://doi.org/10.5281/zenodo.439510>
20. Reger, G., Suda, M.: Checkable proofs for first-order theorem proving. In: Reger, G., Traytel, D. (eds.) 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements (ARCADE 2017). EPICT Series in Computing, vol. 51, pp. 55–63. EasyChair (2017). <https://doi.org/10.29007/s6d1>
21. Zhao, Y., Schmidt, R.A.: FAME: an automated tool for semantic forgetting in expressive description logics. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 19–27. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-94205-6\\_2](https://doi.org/10.1007/978-3-319-94205-6_2)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Actions over Core-Closed Knowledge Bases

Claudia Cauli<sup>1,2(✉)</sup>, Magdalena Ortiz<sup>3</sup>, and Nir Piterman<sup>1</sup>

<sup>1</sup> University of Gothenburg, Gothenburg, Sweden  
claudiacauli@gmail.com

<sup>2</sup> Amazon Web Services, Seattle, USA

<sup>3</sup> TU Wien, Vienna, Austria

**Abstract.** We present new results on the application of semantic- and knowledge-based reasoning techniques to the analysis of cloud deployments. In particular, to the security of *Infrastructure as Code* configuration files, encoded as description logic knowledge bases. We introduce an action language to model *mutating actions*; that is, actions that change the structural configuration of a given deployment by adding, modifying, or deleting resources. We mainly focus on two problems: the problem of determining whether the execution of an action, no matter the parameters passed to it, will not cause the violation of some security requirement (*static verification*), and the problem of finding sequences of actions that would lead the deployment to a state where (un)desirable properties are (not) satisfied (*plan existence* and *plan synthesis*). For all these problems, we provide definitions, complexity results, and decision procedures.

## 1 Introduction

The use of automated reasoning techniques to analyze the properties of cloud infrastructure is gaining increasing attention [4–7, 18]. Despite that, more effort needs to be put into the modeling and verification of generic security requirements over cloud infrastructure pre-deployment. The availability of formal techniques, providing strong security guarantees, would assist complex system-level analyses such as threat modeling and data flow, which now require considerable time, manual intervention, and expert domain knowledge.

We continue our research on the application of semantic-based and knowledge-based reasoning techniques to cloud deployment *Infrastructure as Code* configuration files. In [14], we reported on our experience using expressive description logics to model and reason about Amazon Web Services’ proprietary Infrastructure as Code framework (AWS CloudFormation). We used the rich constructs of these logics to encode domain knowledge, simulate closed-world reasoning, and express mitigations and exposures to security threats. Due to the high complexity of basic tasks [3, 26], we found reasoning in such a framework to be not efficient at cloud scale. In [15], we introduced *core-closed knowledge*

---

C. Cauli—This work was done prior to joining Amazon.

© The Author(s) 2022

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 281–299, 2022.

[https://doi.org/10.1007/978-3-031-10769-6\\_17](https://doi.org/10.1007/978-3-031-10769-6_17)

*bases*—a lightweight description logic combining closed- and open-world reasoning that is tailored to model cloud infrastructure and efficiently query its security properties. Core-closed knowledge bases enable partially-closed predicates whose interpretation is closed over a *core* part of the knowledge base but open elsewhere. To encode potential exposure to security threats, we studied the query satisfiability problem and (together with the usual query entailment problem) applied it to a new class of conjunctive queries that we called MUST/MAY queries. We were able to answer such queries over core-closed knowledge bases in LOGSPACE in data complexity and NP in combined complexity, improving the required NEXPTIME complexity for satisfiability over *ALCOIQ* (used in [14]).

Here, we enhance the quality of the analyses done over pre-deployment artifacts, giving users and practitioners additional precise insights on the impact of potential changes, fixes, and general improvements to their cloud projects. We enrich core-closed knowledge bases with the notion of *core-completeness*, which is needed to ensure that updates are consistent. We define the syntax and semantics of an action language that is expressive enough to encode *mutating* API calls, i.e., operations that change a cloud deployment configuration by creating, modifying, or deleting existing resources. As part of our effort to improve the quality of automated analysis, we also provide relevant reasoning tools to identify and predict the consequences of these changes. To this end, we consider procedures that determine whether the execution of a mutating action always preserves given properties (*static verification*); determine whether there exists a sequence of operations that would lead a deployment to a configuration meeting certain requirements (*plan existence*); and find such sequences of operations (*plan synthesis*).

The paper is organized as follows. In Sect. 2, we provide background on core-closed knowledge bases, conjunctive queries, and MUST/MAY queries. In Sect. 3, we motivate and introduce the notion of *core-completeness*. In Sect. 4, we define the action language. In Sect. 5, we describe the static verification problem and characterize its complexity. In Sect. 6, we address the planning problem and concentrate on the synthesis of minimal plans satisfying a given requirement expressed using MUST/MAY queries. We discuss related works in Sect. 7 and conclude in Sect. 8. Results and proofs that are omitted in this paper are found in the full version [16].

## 2 Background

Description logics (DLs) are a family of logics for encoding knowledge in terms of concepts, roles, and individuals; analogous to first-order logic unary predicates, binary predicates, and constants, respectively. Standard DL knowledge bases (KBs) have a set of axioms, called *TBox*, and a set of assertions, called *ABox*. The TBox contains axioms that relate to concepts and roles. The ABox contains assertions that relate individuals to concepts and pairs of individuals to roles. KBs are usually interpreted under the open-world assumption, meaning that the asserted facts are not assumed to be complete.

*Core-Closed Knowledge Bases.* In [15], we introduced core-closed knowledge bases (ccKBs) as a suitable description logic formalism to encode cloud deployments. The main characteristic of ccKBs is to allow for a combination of open- and closed-world reasoning that ensures tractability. A DL-Lite<sup>ℱ</sup> ccKB is the tuple  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$  built from the standard knowledge base  $\langle \mathcal{T}, \mathcal{A} \rangle$  and the *core* system  $\langle \mathcal{S}, \mathcal{M} \rangle$ . The former encodes incomplete terminological and assertional knowledge. The latter is, in turn, composed of two parts:  $\mathcal{S}$  (also called the *SBox*), containing axioms that encode the core structural specifications, and  $\mathcal{M}$  (also called the *MBox*), containing positive concept and role assertions that encode the core configuration. Syntactically,  $\mathcal{M}$  is similar to an ABox but, semantically, is assumed to be complete with respect to the specifications in  $\mathcal{S}$ .

The ccKB  $\mathcal{K}$  is defined over the alphabets  $\mathbf{C}$  (of concepts),  $\mathbf{R}$  (of roles), and  $\mathbf{I}$  (of individuals), all partitioned into an open subset and a partially-closed subset. That is, the set of concepts is partitioned into the open concepts  $\mathbf{C}^{\mathcal{K}}$  and the closed (specification) concepts  $\mathbf{C}^{\mathcal{S}}$ ; the set of roles is partitioned into open roles  $\mathbf{R}^{\mathcal{K}}$  and closed (specification) roles  $\mathbf{R}^{\mathcal{S}}$ ; and the set of individuals is partitioned into open individuals  $\mathbf{I}^{\mathcal{K}}$  and closed (model) individuals  $\mathbf{I}^{\mathcal{M}}$ . We call  $\mathbf{C}^{\mathcal{S}}$  and  $\mathbf{R}^{\mathcal{S}}$  core-closed predicates, or partially-closed predicates, as their extension is closed over the core domain  $\mathbf{I}^{\mathcal{M}}$  and open otherwise. In contrast, we call  $\mathbf{C}^{\mathcal{K}}$  and  $\mathbf{R}^{\mathcal{K}}$  open predicates. The syntax of concept and role expressions in DL-Lite<sup>ℱ</sup> [2, 8] is as follows:

$$B ::= \perp \mid A \mid \exists p$$

where  $A$  denotes a concept name and  $p$  is either a role name  $r$  or its inverse  $r^-$ . The syntax of axioms provides for the three following axioms:

$$B^1 \sqsubseteq B^2, \quad B^1 \sqsubseteq \neg B^2, \quad (\text{funct } p),$$

respectively called: *positive inclusion* axioms, *negative inclusion* axioms, and *functionality* axioms. These axioms are contained in the sets  $\mathcal{S}$  and  $\mathcal{T}$ . To precisely denote the subsets of  $\mathcal{S}$  and  $\mathcal{T}$  having only axioms of a given type we use the notation  $PI_{\mathcal{X}}$ ,  $NI_{\mathcal{X}}$ , and  $F_{\mathcal{X}}$ , for  $\mathcal{X} \in \{\mathcal{S}, \mathcal{T}\}$ , which respectively contain only positive inclusion axioms, negative inclusion axioms, and functionality axioms. From now on, we denote symbols from the alphabet  $\mathbf{X}^{\mathcal{X}}$  with the subscript  $\mathcal{X}$ , and symbols from the generic alphabet  $\mathbf{X}$  with no subscript. In core-closed knowledge bases, axioms and assertions fall into the scope of a different set depending on the predicates and individuals that they refer to, according to the set definitions below.

$$\begin{aligned} \mathcal{M} &\subseteq \{A_{\mathcal{S}}(a_{\mathcal{M}}), R_{\mathcal{S}}(a_{\mathcal{M}}, a), R_{\mathcal{S}}(a, a_{\mathcal{M}})\} \\ \mathcal{A} &\subseteq \{A_{\mathcal{K}}(a_{\mathcal{K}}), R_{\mathcal{K}}(a_{\mathcal{K}}, b_{\mathcal{K}}), A_{\mathcal{S}}(a_{\mathcal{K}}), R_{\mathcal{S}}(a_{\mathcal{K}}, b_{\mathcal{K}})\} \\ \mathcal{S} &\subseteq \{B_{\mathcal{S}}^1 \sqsubseteq B_{\mathcal{S}}^2, B_{\mathcal{S}}^1 \sqsubseteq \neg B_{\mathcal{S}}^2, \text{Func}(\mathcal{P}_{\mathcal{S}})\} \\ \mathcal{T} &\subseteq \{B^1 \sqsubseteq B_{\mathcal{K}}^2, B^1 \sqsubseteq \neg B_{\mathcal{K}}^2, \text{Func}(\mathcal{P}_{\mathcal{K}})\} \end{aligned}$$

In the above definition of the set  $\mathcal{M}$ , role assertions link at least one individual from the core domain  $\mathbf{I}^{\mathcal{M}}$  (denoted as  $a_{\mathcal{M}}$ ) to one individual from the general set

$\mathbf{I}$  (denoted as  $a$ ). Node  $a$  could either be an individual from the open partition  $\mathbf{I}^{\mathcal{K}}$  or the closed partition  $\mathbf{I}^{\mathcal{M}}$ . When  $a$  is an element from the set  $\mathbf{I}^{\mathcal{K}}$ , we refer to it as a “boundary node”, as it sits at the boundary between the core and the open parts of the knowledge base. As mentioned earlier,  $\mathcal{M}$ -assertions are assumed to be complete and consistent with respect to the terminological knowledge given in  $\mathcal{S}$ ; whereas the usual open-world assumption is made for  $\mathcal{A}$ -assertions. The semantics of a DL-Lite $^{\mathcal{F}}$  core-closed KB is given in terms of interpretations  $\mathcal{I}$ , consisting of a non-empty domain  $\Delta^{\mathcal{I}}$  and an interpretation function  $\cdot^{\mathcal{I}}$ . The latter assigns to each concept  $A$  a subset  $A^{\mathcal{I}}$  of  $\Delta^{\mathcal{I}}$ , to each role  $r$  a subset  $r^{\mathcal{I}}$  of  $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ , and to each individual  $a$  a node  $a^{\mathcal{I}}$  in  $\Delta^{\mathcal{I}}$ , and it is extended to concept expressions in the usual way. An interpretation  $\mathcal{I}$  is a model of an inclusion axiom  $B_1 \sqsubseteq B_2$  if  $B_1^{\mathcal{I}} \subseteq B_2^{\mathcal{I}}$ . An interpretation  $\mathcal{I}$  is a model of a membership assertion  $A(a)$ , (resp.  $r(a, b)$ ) if  $a^{\mathcal{I}} \in A^{\mathcal{I}}$  (resp.  $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ ). We say that  $\mathcal{I}$  models  $\mathcal{T}$ ,  $\mathcal{S}$ , and  $\mathcal{A}$  if it models all axioms or assertions contained therein. We say that  $\mathcal{I}$  models  $\mathcal{M}$ , denoted  $\mathcal{I} \models^{\text{CWA}} \mathcal{M}$ , when it models an  $\mathcal{M}$ -assertion  $f$  if and only if  $f \in \mathcal{M}$ . Finally,  $\mathcal{I}$  models  $\mathcal{K}$  if it models  $\mathcal{T}$ ,  $\mathcal{S}$ ,  $\mathcal{A}$ , and  $\mathcal{M}$ . When  $\mathcal{K}$  has at least one model, we say that  $\mathcal{K}$  is satisfiable.

In the remainder of this paper, we will sometimes refer to the *lts* interpretation of  $\mathcal{M}$ . The *lts* interpretation of  $\mathcal{M}$ , denoted  $lts(\mathcal{M})$ , is the interpretation  $(\Delta^{lts(\mathcal{M})}, \cdot^{lts(\mathcal{M})})$  defined only over concept and role names from the set  $\mathbf{C}^{\mathcal{S}}$  and  $\mathbf{R}^{\mathcal{S}}$ , respectively, and over individual names from  $\mathbf{I}^{\mathcal{K}}$  that appear in the scope of  $\mathcal{M}$ -assertions. The interpretation  $lts(\mathcal{M})$  is the *unique* model of  $\mathcal{M}$  such that  $lts(\mathcal{M}) \models^{\text{CWA}} \mathcal{M}$ .

In the application presented in [14], description logic KBs are used to encode machine-readable deployment files containing multiple resource declarations. Every resource declaration has an underlying tree structure, whose leaves can potentially link to the roots of other resource declarations. Let  $\mathbf{I}^r \subseteq \mathbf{I}^{\mathcal{M}}$  be the set of all resource nodes, we encode their resource declarations in  $\mathcal{M}$ , and formalize the resulting forest structure by partitioning  $\mathcal{M}$  into multiple subsets  $\{\mathcal{M}_i\}_{i \in \mathbf{I}^r}$ , each representing a tree of assertions rooted at a resource node  $i$  (we generally refer to constants in  $\mathcal{M}$  as nodes). For the purpose of this work, we will refer to core-closed knowledge bases where  $\mathcal{M}$  is partitioned as described; that is, ccKBs such that  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \{\mathcal{M}_i\}_{i \in \mathbf{I}^r} \rangle$ .

*Conjunctive Queries.* A *conjunctive query* (CQ) is an existentially-quantified formula  $q[\vec{x}]$  of the form  $\exists \vec{y}. conj(\vec{x}, \vec{y})$ , where *conj* is a conjunction of positive atoms and potentially inequalities. A *union of conjunctive queries* (UCQ) is a disjunction of CQs. The variables in  $\vec{x}$  are called *answer variables*, those in  $\vec{y}$  are the existentially-quantified *query variables*. A tuple  $\vec{c}$  of constants appearing in the knowledge base  $\mathcal{K}$  is an answer to  $q$  if for all interpretations  $\mathcal{I}$  model of  $\mathcal{K}$  we have  $\mathcal{I} \models q[\vec{c}]$ . We call these tuples the *certain answers* of  $q$  over  $\mathcal{K}$ , denoted  $ans(\mathcal{K}, q)$ , and the problem of testing whether a tuple is a certain answer *query entailment*. A tuple  $\vec{c}$  of constants appearing in  $\mathcal{K}$  satisfies  $q$  if there exists an interpretation  $\mathcal{I}$  model of  $\mathcal{K}$  such that  $\mathcal{I} \models q[\vec{c}]$ . We call these tuples the *sat answers* of  $q$  over  $\mathcal{K}$ , denoted  $sat-ans(\mathcal{K}, q)$ , and the problem of testing whether a given tuple is a sat answer *query satisfiability*.

**MUST/MAY Queries.** A MUST/MAY query  $\psi$  [15] is a Boolean combination of nested UCQs in the scope of a MUST or a MAY operator as follows:

$$\psi ::= \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \text{MUST } \varphi \mid \text{MAY } \varphi_{\neq}$$

where  $\varphi$  and  $\varphi_{\neq}$  are unions of conjunctive queries potentially containing inequalities. The reasoning needed for answering the nested queries can be decoupled from the reasoning needed to answer the higher-level formula: nested queries  $\text{MUST } \varphi$  are reduced to conjunctive query entailment, and nested queries  $\text{MAY } \varphi_{\neq}$  are reduced to conjunctive query satisfiability. We denote by  $\text{ANS}(\psi, \mathcal{K})$  the answers of a MUST/MAY query  $\psi$  over the core-closed knowledge base  $\mathcal{K}$ .

### 3 Core-Complete Knowledge Bases

The algorithm **Consistent** presented in [15] computes satisfiability of DL-Lite <sup>$\mathcal{F}$</sup>  core-closed knowledge bases relying on the assumption that  $\mathcal{M}$  is complete and consistent with respect to  $\mathcal{S}$ . Such an assumption effectively means that the information contained in  $\mathcal{M}$  is *explicitly* present and *cannot be completed by inference*. The algorithm relies on the existence of a theoretical object, the canonical interpretation, in which missing assertions can always be introduced when they are logically implied by the positive inclusion axioms. As a matter of fact, positive inclusion axioms are not even included in the inconsistency formula built for the satisfiability check, as it is proven that the canonical interpretation always satisfies them ([15], Lemma 3). When the assumption that  $\mathcal{M}$  is consistent with respect to  $\mathcal{S}$  is dropped, the algorithm **Consistent** becomes insufficient to check satisfiability. We illustrate this with an example.

*Example 1 (Required Configuration).* Let us consider the axioms constraining the AWS resource type `S3::Bucket`. In particular, the  $\mathcal{S}$ -axiom `S3::Bucket`  $\sqsubseteq \exists \text{loggingConfiguration}$  prescribing that all buckets must have a *required* logging configuration. For a set  $\mathcal{M} = \{\text{S3::Bucket}(b)\}$ , according to the partially-closed semantics of core-closed knowledge bases, the absence of an assertion `loggingConfiguration(b, x)`, for some  $x$ , is interpreted as the assertion being false in  $\mathcal{M}$ , which is therefore not consistent with respect to  $\mathcal{S}$ . However, the algorithm **Consistent** will check the *lts* interpretation of  $\mathcal{M}$  for an empty formula (as there are no negative inclusion or functionality axioms) and return *true*.

In essence, the algorithm **Consistent** does not compute the full satisfiability of the whole core-closed knowledge base, but only of its open part. Satisfiability of  $\mathcal{M}$  with respect to the positive inclusion axioms in  $\mathcal{S}$  needs to be checked separately. We introduce a new notion to denote when a set  $\mathcal{M}$  is complete with respect to  $\mathcal{S}$  that is distinct from the notion of consistency. Let  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$  be a DL-Lite <sup>$\mathcal{F}$</sup>  core-closed knowledge base; we say that  $\mathcal{K}$  is *core-complete* when  $\mathcal{M}$  models *all* positive inclusion axioms in  $\mathcal{S}$  under a closed-world assumption; we say that  $\mathcal{K}$  is *open-consistent* when  $\mathcal{M}$  and  $\mathcal{A}$  model all negative inclusion and functionality axioms in  $\mathcal{K}$ 's negative inclusion closure. Finally, we say that  $\mathcal{K}$  is *fully satisfiable* when is both *core-complete* and *open-consistent*.



**Lemma 1.** *In order to check full satisfiability of a  $DL\text{-}Lite^{\mathcal{F}}$  core-closed KB, one simply needs to check if  $\mathcal{K}$  is core-complete (that is, if  $\mathcal{M}$  models all positive axioms in  $\mathcal{S}$  under a closed-world assumption) and if  $\mathcal{K}$  is open-consistent (that is, to run the algorithm Consistent).*

*Proof.* Dropping the assumption that  $\mathcal{M}$  is consistent w.r.t.  $\mathcal{S}$  causes Lemma 3 from [15] to fail. In particular, the canonical interpretation of  $\mathcal{K}$ ,  $can(\mathcal{K})$ , would still be a model of  $PI_{\mathcal{T}}$ ,  $\mathcal{A}$ , and  $\mathcal{M}$ , but may *not* be a model of  $PI_{\mathcal{S}}$ . This is due to the construction of the canonical model that is based on the notion of applicable axioms. In rules **c5-c8** of [15] Definition 1, axioms in  $PI_{\mathcal{S}}$  are defined as applicable to assertions involving open nodes  $a_{\mathcal{K}}$  but *not* to model nodes  $a_{\mathcal{M}}$  in  $\mathbf{I}^{\mathcal{M}}$ . As a result, if the implications of such axioms on model nodes are not included in  $\mathcal{M}$  itself, then they will not be included in  $can(\mathcal{K})$  either, and  $can(\mathcal{K})$  will not be a model of  $PI_{\mathcal{S}}$ . On the other hand, one can easily verify that Lemmas 1,2,4,5,6,7 and Corollary 1 would still hold as they do not rely on the assumption. However, since it is not guaranteed anymore that  $\mathcal{M}$  satisfies all positive inclusion axioms from  $\mathcal{S}$ , the *if* direction of [15] Theorem 1 does not hold anymore: there can be an unsatisfiable ccKB  $\mathcal{K}$  such that  $db(\mathcal{A}) \cup lts(\mathcal{M}) \models cln(\mathcal{T} \cup \mathcal{S}), \mathcal{A}, \mathcal{M}$ . For instance, the knowledge base from Example 1. We also note that the negative inclusion and functionality axioms from  $\mathcal{S}$  will be checked anyway by the consistency formula, both on  $db(\mathcal{A})$  and on  $lts(\mathcal{M})$ .

**Lemma 2.** *Checking whether a  $DL\text{-}Lite^{\mathcal{F}}$  core-closed knowledge base is core-complete can be done in polynomial time in  $\mathcal{M}$ . As a consequence, checking full satisfiability is also done in polynomial time in  $\mathcal{M}$ .*

*Proof.* One can write an algorithm that checks *core-completeness* by searching for the existence of a positive inclusion axiom  $B_S^1 \sqsubseteq B_S^2 \in PI_{\mathcal{S}}$  such that  $\mathcal{M} \models B_S^1(a_{\mathcal{M}})$  and  $\mathcal{M} \not\models B_S^2(a_{\mathcal{M}})$ , where the relation  $\models$  is defined over  $DL\text{-}Lite^{\mathcal{F}}$  concept expressions as follows:

$$\begin{aligned} \mathcal{M} \models \perp(a_{\mathcal{M}}) &\leftrightarrow \text{false} \\ \mathcal{M} \models A_S(a_{\mathcal{M}}) &\leftrightarrow A_S(a_{\mathcal{M}}) \in \mathcal{M} \\ \mathcal{M} \models \exists r_S(a_{\mathcal{M}}) &\leftrightarrow \exists b. r_S(a_{\mathcal{M}}, b) \in \mathcal{M} \\ \mathcal{M} \models \exists r_S^-(a_{\mathcal{M}}) &\leftrightarrow \exists b. r_S(b, a_{\mathcal{M}}) \in \mathcal{M}. \end{aligned}$$

The knowledge base is *core-complete* if such a node cannot be found.

## 4 Actions

We now introduce a formal language to encode mutating actions. Let us remind ourselves that, in our application of interest, the execution of a mutating action modifies the configuration of a deployment by either adding new resource instances, deleting existing ones, or modifying their settings. Here, we introduce a framework for  $DL\text{-}Lite^{\mathcal{F}}$  core-closed knowledge base updates, triggered by the execution of an action that enables all the above mentioned effects. The

only component of the core-closed knowledge base that is modified by the action execution is  $\mathcal{M}$ ; while  $\mathcal{T}$ ,  $\mathcal{S}$ , and  $\mathcal{A}$  remain unchanged. As a consequence of updating  $\mathcal{M}$ , actions can introduce new individuals and delete old ones, thus updating the set  $\mathbf{I}^{\mathcal{M}}$  as well. Note that this may force changes outside  $\mathbf{I}^{\mathcal{M}}$  due to the axioms in  $\mathcal{T}$  and  $\mathcal{S}$ . The effects of applying an action over  $\mathcal{M}$  depend on a set of input parameters that will be instantiated at execution time, resulting in different assertions being added or removed from  $\mathcal{M}$ . As a consequence of assertions being added, fresh individuals might be introduced in the active domain of  $\mathcal{M}$ , including both model nodes from  $\mathbf{I}^{\mathcal{M}}$  and boundary nodes from  $\mathbf{I}^{\mathcal{B}}$ . Differently, as a consequence of assertions being removed, individuals might be removed from the active domain of  $\mathcal{M}$ , including model nodes from  $\mathbf{I}^{\mathcal{M}}$  but *not* including boundary nodes from  $\mathbf{I}^{\mathcal{B}}$ . In fact, boundary nodes are owned by the open portion of the knowledge base and are known to exist regardless of them being used in  $\mathcal{M}$ . We invite the reader to review the set definitions for  $\mathcal{A}$ - and  $\mathcal{M}$ -assertions (Sect. 2) to note that it is indeed possible for a generic boundary individual  $a$  involved in an  $\mathcal{M}$ -assertion to also be involved in an  $\mathcal{A}$ -assertion.

#### 4.1 Syntax

An action is defined by a signature and a body. The signature consists of an action name and a list of formal parameters, which will be replaced with actual parameters at execution time. The body, or action effect, can include conditional statements and concatenation of atomic operations over  $\mathcal{M}$ -assertions. For example, let  $\alpha$  be the action  $act(\vec{x}) = \gamma$ ; that is, the action denoted by signature  $act(\vec{x})$  and body  $\gamma$ , with signature name  $act$ , signature parameters  $\vec{x}$ , and body effect  $\gamma$ . Since it contains unbound parameters, or free variables, action  $\alpha$  is ungrounded and needs to be instantiated with actual values in order to be executed over a set  $\mathcal{M}$ . In the following, we assume the existence of a set  $\mathbf{Var}$ , of variable names, and consider a generic input parameters substitution  $\vec{\theta} : \mathbf{Var} \rightarrow \mathbf{I}$ , which replaces each variable name by an individual node. For simplicity, we will denote an ungrounded action by its effect  $\gamma$ , and a grounded action by the composition of its effect with an input parameter substitution  $\gamma\vec{\theta}$ . Action effects can either be *complex* or *basic*. The syntax of complex action effects  $\gamma$  and basic effects  $\beta$  is constrained by the following grammar.

$$\begin{aligned} \gamma &::= \epsilon \mid \beta \cdot \gamma \mid [\varphi \rightsquigarrow \beta] \cdot \gamma \\ \beta &::= \oplus_x S \mid \ominus_x S \mid \odot_{x_{new}} S \mid \ominus_x \end{aligned}$$

The complex action effects  $\gamma$  include: the empty effect ( $\epsilon$ ), the execution of a basic effect followed by a complex one ( $\beta \cdot \gamma$ ), and the conditional execution of a basic effect upon evaluation of a formula  $\varphi$  over the set  $\mathcal{M}$  ( $[\varphi \rightsquigarrow \beta] \cdot \gamma$ ). The basic action effects  $\beta$  include: the addition of a set  $S$  of  $\mathcal{M}$ -assertions to the subset  $\mathcal{M}_x$  ( $\oplus_x S$ ), the removal of a set  $S$  of  $\mathcal{M}$ -assertions from the subset  $\mathcal{M}_x$  ( $\ominus_x S$ ), the addition of a fresh subset  $\mathcal{M}_{x_{new}}$  containing all the  $\mathcal{M}$ -assertions in the set  $S$  ( $\odot_{x_{new}} S$ ), and the removal of an existing  $\mathcal{M}_x$  subset in its entirety ( $\ominus_x$ ). The set  $S$ , the formula  $\varphi$ , and the operators  $\oplus/\ominus$  might contain *free*

*variables*. These variables are of two types: (1) variables that are replaced by the grounding of the action input parameters, and (2) variables that are the answer variables of the formula  $\varphi$  and appear in the nested effect  $\beta$ .

*Example 2.* The following is the definition of the action `createBucket` from the API reference of the AWS resource type `S3::Bucket`. The input parameters are two: the new bucket name “*name*” and the canned access control list “*acl*” (one of *Private*, *PublicRead*, *PublicReadWrite*, *AuthenticatedRead*, etc.). The effect of the action is to add a fresh subset  $\mathcal{M}_x$  for the newly introduced individual  $x$  containing the two assertions `S3::Bucket( $x$ )` and `accessControl( $x, y$ )`.

$$\text{createBucket}(x : \text{name}, y : \text{acl}) = \odot_x \{ \text{S3::Bucket}(x), \text{accessControl}(x, y) \} \cdot \epsilon$$

The action needs to be instantiated by a specific parameter assignment, for example the substitution  $\theta = [x \leftarrow \text{DataBucket}, y \leftarrow \text{Private}]$ , which binds the variable  $x$  to the node `DataBucket` and the variable  $y$  to the node `Private`, both taken from a pool of inactive nodes in **I**.

*Action Query  $\varphi$ .* The syntax introduced in the previous paragraph allows for complex actions that conditionally execute a basic effect  $\beta$  depending on the evaluation of a formula  $\varphi$  over  $\mathcal{M}$ . This is done via the construct  $[\varphi \rightsquigarrow \beta] \cdot \gamma$ . The formula  $\varphi$  might have a set  $\vec{y}$  of answer variables that appear free in its body and are then bound to concrete tuples of nodes during evaluation. The answer tuples are in turn used to instantiate the free variables in the nested effect  $\beta$ . We call  $\varphi$  the *action query* since we use it to select all the nodes that will be involved in the action effect. According to the grammar below,  $\varphi$  is a boolean combination of  $\mathcal{M}$ -assertions potentially containing free variables.

$$\varphi ::= A_S(t) \mid R_S(t_1, t_2) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_2 \vee \varphi_2 \mid \neg \varphi$$

In particular,  $A_S$  is a symbol from the set  $\mathbf{C}^S$  of partially-closed concepts;  $R_S$  is a symbol from the set  $\mathbf{R}^S$  of partially-closed roles; and  $t, t_1, t_2$  are either individual or variable names from the set  $\mathbf{I} \uplus \mathbf{Var}$ , chosen in such a way that the resulting assertion is an  $\mathcal{M}$ -assertion. Since the formula  $\varphi$  can only refer to  $\mathcal{M}$ -assertions, which are interpreted under a closed semantics, its evaluation requires looking at the content of the set  $\mathcal{M}$ . A formula  $\varphi$  with no free variables is a boolean formula and evaluates to either true or false. A formula  $\varphi$  with answer variables  $\vec{y}$  and arity  $ar(\varphi)$  evaluates to all the tuples  $\vec{t}$ , of size equal the arity of  $\varphi$ , that make the formula true in  $\mathcal{M}$ . The free variables of  $\varphi$  can only appear in the action  $\beta$  such that  $\varphi \rightsquigarrow \beta$ . We denote by  $\mathbf{ANS}(\varphi, \mathcal{M})$  the set of answers to the action query  $\varphi$  over  $\mathcal{M}$ . It is easy to see that the maximum number of tuples that could be returned by the evaluation (that is, the size of the set  $\mathbf{ANS}(\varphi, \mathcal{M})$ ) is bounded by  $|\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^B|^{ar(\varphi)}$ , in turn bounded by  $(2|\mathcal{M}|)^{2|\varphi|}$ .

*Example 3.* The following example shows the encoding of the S3 API operation called `deleteBucketEncryption`, which requires as unique input parameter the name of the bucket whose encryption configuration is to be deleted. Since

a bucket can have multiple encryption configuration rules (each prescribing different encryption keys and algorithms to be used) we use an action query  $\varphi$  to select *all* the nodes that match the assertions structure to be removed.

$$\varphi[y, k, z](x) = \text{S3::Bucket}(x) \wedge \text{encrRule}(x, y) \wedge \text{SSEKey}(y, k) \wedge \text{SSEAlgo}(y, z)$$

The query  $\varphi$  is instantiated by the specific bucket instance (which will replace the variable  $x$ ) and returns all the triples  $(y, k, z)$  of encryption rule, key, and algorithm, respectively, which identify the assertions corresponding to the different encryption configurations that the bucket has. The answer variables are then used in the action effect to instantiate the assertions to remove from  $\mathcal{M}_x$ :

$$\begin{aligned} & \text{deleteBucketEncryption}(x : \text{name}) \\ & = [\varphi[y, k, z](x) \rightsquigarrow \ominus_x \{\text{encrRule}(x, y), \text{SSEKey}(y, k), \text{SSEAlgo}(y, z)\}] \cdot \epsilon \end{aligned}$$

## 4.2 Semantics

So far, we have described the syntax of our action language and provided two examples that showcase the encoding of real-world API calls. Now, we define the semantics of action effects with respect to the changes that they induce over a knowledge base. Let us recall that given a substitution  $\vec{\theta}$  for the input parameters of an action  $\gamma$ , we denote by  $\gamma\vec{\theta}$  the grounded action where all the input variables are replaced according to what prescribed by  $\vec{\theta}$ . Let us also recall that the effects of an action apply only to assertions in  $\mathcal{M}$  and individuals from  $\mathbf{I}^{\mathcal{M}}$ , and cannot affect nodes and assertions from the open portion of the knowledge base.

The execution of a grounded action  $\gamma\vec{\theta}$  over a DL-Lite<sup>F</sup> core-closed knowledge base  $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M})$ , defined over the set  $\mathbf{I}^{\mathcal{M}}$  of partially-closed individuals, generates a new knowledge base  $\mathcal{K}^{\gamma\vec{\theta}} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}^{\gamma\vec{\theta}})$ , defined over an updated set of partially-closed individuals  $\mathbf{I}^{\mathcal{M}^{\gamma\vec{\theta}}}$ . Let  $S$  be a set of  $\mathcal{M}$ -assertions,  $\gamma$  a complex action,  $\vec{\theta}$  an input parameter substitution, and  $\vec{\rho}$  a generic substitution that potentially replaces all free variables in the action  $\gamma$ . Let  $\vec{\rho}_1$  and  $\vec{\rho}_2$  be two substitutions with signature  $\text{Var} \rightarrow \mathbf{I}$  such that  $\text{dom}(\vec{\rho}_1) \cap \text{dom}(\vec{\rho}_2) = \emptyset$ ; we denote their composition by  $\vec{\rho}_1\vec{\rho}_2$  and define it as the new substitution such that  $\vec{\rho}_1\vec{\rho}_2(x) = a$  if  $\vec{\rho}_1(x) = a \vee \vec{\rho}_2(x) = a$ , and  $\vec{\rho}_1\vec{\rho}_2(x) = \perp$  if  $\vec{\rho}_1(x) = \perp \wedge \vec{\rho}_2(x) = \perp$ . We formalize the application of the grounded action  $\gamma\vec{\theta}$  as the transformation  $T_{\gamma\vec{\theta}}$  that maps the pair  $\langle \mathcal{M}, \mathbf{I}^{\mathcal{M}} \rangle$  into the new pair  $\langle \mathcal{M}', \mathbf{I}^{\mathcal{M}'} \rangle$ . We sometimes use the notation  $T_{\gamma\vec{\theta}}(\mathcal{M})$  or  $T_{\gamma\vec{\theta}}(\mathbf{I}^{\mathcal{M}})$  to refer to the updated MBox or to the updated set of model nodes, respectively. The rules for applying the transformation depend on the structure of the action  $\gamma$  and are reported in Fig. 1. The transformation starts with an initial generic substitution  $\vec{\rho} = \vec{\theta}$ . As the transformation progresses, the generic substitution  $\vec{\rho}$  can be updated only as a result of the evaluation of an action query  $\varphi$  over  $\mathcal{M}$ . Precisely, all the tuples  $t_1, \dots, t_n$  making  $\varphi$  true in  $\mathcal{M}$  will be considered and composed with the current substitution  $\vec{\rho}$  generating  $n$  fresh substitutions  $\vec{\rho}t_1, \dots, \vec{\rho}t_n$  which are used in the subsequent application of the nested effect  $\beta$ . Since the core  $\mathcal{M}$  of the knowledge base  $\mathcal{K}$  changes at every

action execution, its domain of model nodes  $\mathbf{I}^{\mathcal{M}}$  changes as well. The execution of an action  $\gamma_{\vec{\theta}}$  over the knowledge base  $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M})$  with set of model nodes  $\mathbf{I}^{\mathcal{M}}$  could generate a new  $\mathcal{K}^{\gamma_{\vec{\theta}}} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}^{\gamma_{\vec{\theta}}})$  with a new set of model nodes  $\mathbf{I}^{\mathcal{M}'}$  that is not *core-complete* or not *open-consistent* (see Sect. 3 for the corresponding definitions). We illustrate two examples next.

$$\begin{aligned}
T_{\epsilon_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\mathcal{M}, \mathbf{I}^{\mathcal{M}}) \\
T_{\beta \cdot \gamma_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= T_{\gamma_{\vec{\rho}}}(T_{\beta_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}})) \\
T_{[\varphi \rightsquigarrow \beta] \cdot \gamma_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= \begin{cases} T_{\gamma_{\vec{\rho}}}(T_{\beta_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}})) & \text{if } \text{ANS}(\varphi, \mathcal{M}) = tt \\ T_{\gamma_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) & \text{if } \text{ANS}(\varphi, \mathcal{M}) = \emptyset \text{ or } ff \\ T_{\gamma_{\vec{\rho}}}(T_{\beta_{\vec{\rho}}\vec{t}_1 \dots \beta_{\vec{\rho}}\vec{t}_n}(\mathcal{M}, \mathbf{I}^{\mathcal{M}})) & \text{if } \text{ANS}(\varphi, \mathcal{M}) = \{\vec{t}_1, \dots, \vec{t}_n\} \end{cases} \\
T_{\oplus_x S_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\{ \mathcal{M}_i \}_{i \neq \vec{\rho}(x)} \cup \{ \mathcal{M}_{\vec{\rho}(x)} \cup S_{\vec{\rho}} \}, \mathbf{I}^{\mathcal{M}} \cup \text{ind}(S_{\vec{\rho}})) \\
T_{\ominus_x S_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\{ \mathcal{M}_i \}_{i \neq \vec{\rho}(x)} \cup \{ \mathcal{M}_{\vec{\rho}(x)} \setminus S_{\vec{\rho}} \}, \mathbf{I}^{\mathcal{M}} \setminus \text{ind}(S_{\vec{\rho}})) \\
T_{\odot_x S_{\vec{\rho}}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\mathcal{M} \cup \{ \mathcal{M}_{\vec{\rho}(x)} = S_{\vec{\rho}} \}, \mathbf{I}^{\mathcal{M}} \cup \text{ind}(S_{\vec{\rho}})) \\
T_{\ominus_x \vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\mathcal{M} \setminus \mathcal{M}_{\vec{\rho}(x)}, \mathbf{I}^{\mathcal{M}} \setminus \text{ind}(\mathcal{M}_{\vec{\rho}(x)}))
\end{aligned}$$

**Fig. 1.** Semantic of the action language defined over the MBox  $\mathcal{M}$  and set  $\mathbf{I}^{\mathcal{M}}$ .

*Example 4 (Violation of core-completeness).* Consider the case where the general specifications of the system require all objects of type bucket to have a logging configuration, and an action that removes the logging configuration from a bucket. Consider the core-closed knowledge base  $\mathcal{K}$  where  $\mathcal{S} = \{\text{S3::Bucket} \sqsubseteq \exists \text{loggingConfiguration}\}$  and  $\mathcal{M} = \{\text{S3::Bucket}(b), \text{loggingConfiguration}(b, c)\}$  (consistent wrt  $\mathcal{S}$ ) and the action  $\gamma$  defined as

$$\begin{aligned}
&\text{deleteLoggingConfiguration}(x : \text{name}) \\
&= [(\varphi[y](x) = \text{S3::Bucket}(x) \wedge \text{loggingConfiguration}(x, y)) \\
&\quad \rightsquigarrow \ominus_x \{\text{loggingConfiguration}(x, y)\}] \cdot \epsilon
\end{aligned}$$

For the input parameter substitution  $\vec{\theta} = [x \leftarrow b]$ , it is easy to see that the transformation  $T_{\gamma_{\vec{\theta}}}$  applied to  $\mathcal{M}$  results in the update  $\mathcal{M}^{\gamma_{\vec{\theta}}} = \{\text{S3::Bucket}(b)\}$ , which is *not* core-complete.

*Example 5 (Violation of open-consistency).* Consider the case where an action application indirectly affects boundary nodes and their properties, leading to inconsistencies in the open portion of the knowledge base. For example, when the knowledge base prescribes that buckets used to store logs cannot be public; however, a change in the configuration of a bucket instance causes a second bucket (initially known to be public) to also become a log store. In particular, this happens when the knowledge base  $\mathcal{K}$  contains the  $\mathcal{T}$ -axiom  $\exists \text{loggingDestination}^- \sqsubseteq \neg \text{PublicBucket}$  and the  $\mathcal{A}$ -assertion  $\text{PublicBucket}(b)$ , and

we apply an action that introduces a new bucket storing its logs to  $b$ , defined as follows:

$$\begin{aligned} & \text{createBucketWithLogging}(x : \text{name}, y : \text{log}) \\ &= \odot_x \{ \text{S3::Bucket}(x), \text{loggingDestination}(x, y) \} \end{aligned}$$

For the input parameter substitution  $\vec{\theta} = [x \leftarrow \text{newBucket}, y \leftarrow b]$ , the result of applying the transformation  $T_{\gamma\vec{\theta}}$  is the set  $\mathcal{M} = \{ \text{S3::Bucket}(\text{newBucket}), \text{loggingDestination}(\text{newBucket}, b) \}$  which, combined with the pre-existing and unchanged sets  $\mathcal{T}$  and  $\mathcal{A}$ , causes the updated  $\mathcal{K}^{\gamma\vec{\theta}}$  to be *not* open-consistent.

From a practical point of view, the examples highlight the need to re-evaluate core-completeness and open-consistency of a core-closed knowledge base after each action execution. Detecting a violation to core-completeness signals that we have modeled an action that is inconsistent with respect to the systems specifications, which most likely means that the action is missing something and needs to be revised. Detecting a violation to open-consistency signals that our action, even when consistent with respect to the specifications, introduces a change that conflicts with other assumptions that we made about the system, and generally indicates that we should either revise the assumptions or forbid the application of the action. Both cases are important to consider in the development life cycle of the core-closed KB and the action definitions.

## 5 Static Verification

In this section, we investigate the problem of computing whether the execution of an action, no matter the specific instantiation, always preserves given properties of core-closed knowledge bases. We focus on properties expressed as MUST/MAY queries and define the static verification problem as follows.

**Definition 1 (Static Verification).** *Let  $\mathcal{K}$  be a DL-Lite<sup>F</sup> core-closed knowledge base,  $q$  be a MUST/MAY query, and  $\gamma$  be an action with free variables from the language presented above. Let  $\vec{\theta}$  be an assignment for the input variables of  $\gamma$  that transforms  $\gamma$  into the grounded action  $\gamma\vec{\theta}$ . Let  $\mathcal{K}^{\gamma\vec{\theta}}$  be the DL-Lite<sup>F</sup> core-closed knowledge base resulting from the application of the grounded action  $\gamma\vec{\theta}$  onto  $\mathcal{K}$ . We say that the action  $\gamma$  “preserves  $q$  over  $\mathcal{K}$ ” iff for every grounded instance  $\gamma\vec{\theta}$  we have that  $\text{ANS}(q, \mathcal{K}) = \text{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}})$ . The static verification problem is that of determining whether an action  $\gamma$  is  $q$ -preserving over  $\mathcal{K}$ .*

An action  $\gamma$  is *not*  $q$ -preserving over  $\mathcal{K}$  iff there exists a grounding  $\vec{\theta}$  for the input variables of  $\gamma$  such that  $\text{ANS}(q, \mathcal{K}) \neq \text{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}})$ ; that is, fixed the grounding  $\vec{\theta}$  there exists a tuple  $\vec{t}$  for  $q$ ’s answer variables such that  $\vec{t} \in \text{ANS}(q, \mathcal{K}) \setminus \text{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}})$  or  $\vec{t} \in \text{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}}) \setminus \text{ANS}(q, \mathcal{K})$ .

**Theorem 1 (Complexity of the Static Verification Problem).** *The static verification problem, i.e. deciding whether an action  $\gamma$  is  $q$ -preserving over  $\mathcal{K}$ , can be decided in PTIME in data complexity and EXPTIME in the arities of  $\gamma$  and  $q$ .*

*Proof.* The proof relies on the fact that one could: enumerate all possible assignments  $\vec{\theta}$ ; compute the updated knowledge bases  $\mathcal{K}^{\gamma\vec{\theta}}$ ; check whether these are fully satisfiable; enumerate all tuples  $\vec{t}$  for the query  $q$ ; and, finally, check whether there exists at least one such tuple that satisfies  $q$  over  $\mathcal{K}$  but not  $\mathcal{K}^{\gamma\vec{\theta}}$  or vice versa. The number of assignments  $\vec{\theta}$  is bounded by  $(|\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}| + ar(\gamma))^{ar(\gamma)}$  as it is sufficient to replace each variable appearing in the action  $\gamma$  either by a known object from  $\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}$  or by a fresh one. The computation of the updated  $\mathcal{K}^{\gamma\vec{\theta}}$  is done in polynomial time in  $\mathcal{M}$  (and is exponential in the size of the action  $\gamma$ ) as it may require the evaluation of an internal action query  $\varphi$  and the consecutive re-application of the transformation for a number of tuples that is bounded by a polynomial over the size of  $\mathcal{M}$ . As explained in Sect. 3, checking full satisfiability of the resulting core-closed knowledge base is also polynomial in  $\mathcal{M}$ . The number of tuples  $\vec{t}$  is bounded by  $(|\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}| + ar(\gamma))^{ar(q)}$  as it is enough to consider all those tuples involving known objects plus the fresh individuals introduced by the assignment  $\vec{\theta}$ . Checking whether a tuple  $\vec{t}$  satisfies the query  $q$  over a core-closed knowledge base is decided in LOGSPACE in the size of  $\mathcal{M}$  [15] which is, thus, also polynomial in  $\mathcal{M}$ .

## 6 Planning

As discussed throughout the paper, the execution of a mutating action modifies the configuration of a deployment and potentially changes its posture with respect to a given set of requirements. In the previous two sections, we introduced a language to encode mutating actions and we investigated the problem of checking whether the application of an action preserves the properties of a core-closed knowledge base. In this section, we investigate the plan existence and synthesis problems; that is, the problem of deciding whether there exists a sequence of grounded actions that leads the knowledge base to a state where a certain requirement is met, and the problem of finding a set of such plans, respectively. We start by defining a notion of transition system that is generated by applying actions to a core-closed knowledge base and then use this notion to focus on the mentioned planning problems. As in classical planning, the plan existence problem for plans computed over unbounded domains is undecidable [17, 19]. The undecidability proof is done via reduction from the Word problem. The problem of deciding whether a deterministic Turing machine  $M$  accepts a word  $w \in \{0, 1\}^*$  is reduced to the plan existence problem. Since undecidability holds even for basic action effects, we can show undecidability over an unbounded domain by using the same encoding of [1].

*Transition Systems.* In the style of the work done in [10, 21], the combination of a DL-Lite <sup>$\mathcal{F}$</sup>  core-closed knowledge base and a set of actions can be viewed as the transition system it generates. Intuitively, the states of the transition system correspond to MBoxes and the transitions between states are labeled by grounded actions. A DL-Lite <sup>$\mathcal{F}$</sup>  core-closed knowledge base  $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$ , defined over the possibly infinite set of individuals  $\mathbf{I}$  (and model nodes  $\mathbf{I}_0^{\mathcal{M}} \subseteq \mathbf{I}$ )

and the set  $\text{Act}$  of ungrounded actions, generates the transition system (TS)  $\mathcal{T}_{\mathcal{K}} = (\mathbf{I}, \mathcal{T}, \mathcal{A}, \mathcal{S}, \Sigma, \mathcal{M}_0, \rightarrow)$  where  $\Sigma$  is a set of *fully satisfiable* (i.e., *core-complete* and *open-consistent*) MBoxes;  $\mathcal{M}_0$  is the initial MBox; and  $\rightarrow \subseteq \Sigma \times L_{\text{Act}} \times \Sigma$  is a labeled transition relation with  $L_{\text{Act}}$  the set of all possible *grounded actions*. The sets  $\Sigma$  and  $\rightarrow$  are defined by mutual induction as the smallest sets such that: if  $\mathcal{M}_i \in \Sigma$  then for every grounded action  $\gamma\vec{\theta} \in L_{\text{Act}}$  such that the fresh MBox  $\mathcal{M}_{i+1}$  resulting from the transformation  $T_{\gamma\vec{\theta}}$  is core-complete and open-consistent, we have that  $\mathcal{M}_{i+1} \in \Sigma$  and  $(\mathcal{M}_i, \gamma\vec{\theta}, \mathcal{M}_{i+1}) \in \rightarrow$ .

Since we assume that actions have input parameters that are replaced during execution by values from  $\mathbf{I}$ , which contains both known objects from  $\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}$  and possibly infinitely many fresh objects, the generated transition system  $\mathcal{T}_{\mathcal{K}}$  is generally infinite. To keep the planning problem decidable, we concentrate on a known finite subset  $\mathcal{D} \subset \mathbf{I}$  containing all the fresh nodes and value assignments to action variables that are of interest for our application. In the remainder of this paper, we discuss the plan existence and synthesis problem for finite transition systems  $\mathcal{T}_{\mathcal{K}} = (\mathcal{D}, \mathcal{T}, \mathcal{A}, \mathcal{S}, \Sigma, \mathcal{M}_0, \rightarrow)$ , whose states in  $\Sigma$  have a domain that is also bounded by  $\mathcal{D}$ .

*The Plan Existence Problem.* A plan is a sequence of grounded actions whose execution leads to a state satisfying a given property. Let  $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$  be a DL-Lite<sup>F</sup> core-closed knowledge base;  $\text{Act}$  be a set of ungrounded actions; and let  $\mathcal{T}_{\mathcal{K}} = (\mathcal{D}, \mathcal{T}, \mathcal{A}, \mathcal{S}, \Sigma, \mathcal{M}_0, \rightarrow)$  be its generated finite TS. Let  $\pi$  be a finite sequence  $\gamma_1\vec{\theta}_1 \cdots \gamma_n\vec{\theta}_n$  of grounded actions taken from the set  $L_{\text{Act}}$ . We call the sequence  $\pi$  *consistent* iff there exists a run  $\rho = \mathcal{M}_0 \xrightarrow{\gamma_1\vec{\theta}_1} \mathcal{M}_1 \xrightarrow{\gamma_2\vec{\theta}_2} \cdots \xrightarrow{\gamma_n\vec{\theta}_n} \mathcal{M}_n$  in  $\mathcal{T}_{\mathcal{K}}$ . Let  $q$  be a MUST/MAY query mentioning objects from  $\text{adom}(\mathcal{K})$  and  $\vec{t}$  a tuple from the set  $\text{adom}(\mathcal{K})^{\text{ar}(q)}$ . A consistent sequence  $\pi$  of grounded actions is a *plan* from  $\mathcal{K}$  to  $(\vec{t}, q)$  iff  $\vec{t} \in \text{ANS}(q, \mathcal{K}_n = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_n))$  with  $\mathcal{M}_n$  the final state of the run induced by  $\pi$ .

**Definition 2 (Plan Existence).** *Given a DL-Lite<sup>F</sup> core-closed knowledge base  $\mathcal{K}$ , a tuple  $\vec{t}$ , and a MUST/MAY query  $q$ , the plan existence problem is that of deciding whether there exists a plan from  $\mathcal{K}$  to  $(\vec{t}, q)$ .*

*Example 6.* Let us consider the transition system  $\mathcal{T}_{\mathcal{K}}$  generated by the core-closed knowledge base  $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$  having the set of partially-closed assertions  $\mathcal{M}_0$  defined as

$$\{\text{S3::Bucket}(b), \text{KMS::Key}(k), \text{bucketEncryptionRule}(b, r), \text{bucketKey}(r, k), \\ \text{bucketKeyEnabled}(r, \text{true}), \text{enableKeyRotation}(k, \text{false})\}$$

and the set of action labels  $\text{Act}$  containing the actions `deleteBucket`, `createBucket`, `deleteKey`, `createKey`, `enableKeyRotation`, `putBucketEncryption`, and `deleteBucketEncryption`. Let us assume that we are interested in verifying the existence of a sequence of grounded actions that when applied onto the knowledge base would configure the bucket node  $b$  to be encrypted with a rotating key. Formally, this is equivalent to checking the existence of a consistent plan  $\pi$  that when executed